Direct Surface Rendering of General and Genetically Bred Implicit Surfaces

M. W. Jones

Computer Science Department, University of Wales Swansea
Singleton Park, Swansea SA2 8PP

Abstract

Genetic algorithms are proposed as a method for creating objects within the volume graphics environment. An approach to the realistic rendering of general volume data is given in the paper, along with example renderings of scanned and genetically produced volume graphic objects. The rendering method is demonstrated by rendering complex implicitly defined surfaces. A description of the genetic operators for the breeding of objects is given, and the representation used for the objects. It is shown that such a method will allow the user to create new models in an easy and intuitive manner by combining existing objects in new and unusual ways. This work complements other research into volume graphics that is currently being carried out in Swansea.

1 Introduction

The creation of new and interesting objects is a difficult process, often manual and labour intensive. Approaches have relied on the expert manipulation of control points for surface patches, artistic understanding in the design of triangular meshes, or even the building and then digitisation of particular objects. Most approaches have in common the fact that the user must be familiar with the outcome they are trying to achieve. They are of limited value to those who just wish to experiment with objects to explore what possibilities are available.

Recently Glassner [1] has addressed this problem by producing a rapid development tool which is based on concepts used within sound synthesis. The user has control over a number of functional units each of which accept inputs and produce outputs. Each unit may be plugged into others in a variety of combinations in order to transform an initial model into the desired output. The work described in this paper has a similar goal – allowing flexible manipulation of objects, but achieves it in a very different way. *Genetic algorithms* [2] are chosen as a means of producing a new generation of sample objects from a previous generation, relying on user interaction as the selection metric. *Volume data* is chosen as a quick route for the *rendering* of the objects – a possibly surprising choice and one that will be described in more detail in section 2.

This work also draws upon the ideas and work presented by Thornborrow [3] at the 1995 UK Eurographics Conference.

The choice of volume data for rendering also has a foundation in the fact that Swansea has a graphics research group that is actively seeking practical methods for realising the potential of *Volume Graphics*. Recent work [4] has shown that many aspects of rendering can be achieved just as well using the volume graphic environment. The main advantage of the

method is that implicitly defined surfaces can be rendered without resorting to numerical techniques or to other forms of visualisation [5].

Creating surfaces within the volume graphics environment may be regarded as more difficult than that of producing surfaces for the traditional graphics approach, and yet work has been published on voxelisation [6–8] and volume sculpting [9]. Usually volume data is the result of some acquisition process such as CT, or the result of some simulation process such as CFD. Any realistic attempt at demonstrating volume graphics must include methods for data production, such as voxelisation or the approach described in this paper.

In addition to describing the rendering method (section 2), the genetic operators used are described in section 3, their implementation in section 4 and results and conclusions are given in section 5.

2 Object Representation and Rendering

Volume data has been the matter of much study since CT and MRI scans were first used by the medical profession in the early '70s. Kajiya [10] was one of the first researchers to indicate that it may be possible to achieve all forms of rendering within the domain of the volume data type. His primary thought was that volume rendering would allow a better scalability for more complex scenes. Various attempts have been made to convert surfaces into volume data via a process known as *voxelisation*. It was demonstrated by Jones [6] that this process is not only feasible, but it also results in faster rendering times for even simple data sets.

The voxelisation process requires that a three dimensional functional representation for an object exists in the from $v = f(x, y, z)$. Each voxel value is computed according to the function f, and the resulting volume can be rendered using traditional techniques [11–13], or by using a surface extractor [14, 15] with a threshold value of, for example, zero. It can be seen from the description, that when the functional description exists, voxelisation is simply the evaluation of that function over the size of the data set.

For the purpose of this work, the surfaces that are considered are exactly those that can be represented implicitly. For example, the torus (figure 1), can be represented by the equation $(x^2 + y^2 + z^2 - (a^2 + b^2))^2 = 4a^2(b^2 - z^2)$. This can be converted to volume data by evaluating the function on a three-dimensional grid of points. In the case of figure 1, the grid is of dimensions $40 \times 40 \times 40$. The method of *Direct Surface Rendering* is then used to create the visualisation of the volume object.

Figure 1: Direct Surface Rendering of a torus volume graphic.

The algorithm bears a resemblance to the direct volume rendering method in principle, but is only interested in the surface contained in the volume, rather than the intensity and colour of every voxel. For each pixel in the image plane, a ray is cast into the volume and is traced through cubes on its path until it hits a transverse cube. A cube, bounded by eight voxels, is said to be transverse if at least one of its voxels is inside the surface and one outside the surface, where the surface is defined as an isosurface of value τ. Transverse cubes can be

identified by comparing values of its bounding voxels against the threshold value and setting an eight bit flag, $b_8b_7b_6b_5b_4b_3b_2b_1$, as:

$$b_i = \begin{cases} 0 & \text{if the value of voxel } i \leq \tau \\ 1 & \text{if the value of voxel } i > \tau \end{cases} \tag{1}$$

A cube is transverse if its flag is not of the binary values 0 or 11111111. Once a transverse cube is found, a further check is made to see if the ray intersects the surface contained within the cube. Given the ray entry and exit points, namely α and β respectively, the function values at those points are calculated using an interpolation function $F(\delta)$. Given a point δ lying on one of the six square facets of a cube, $F(\delta)$ is defined as the bilinear interpolation of the values associated to the four voxels which bound the facet. The ray intersects the surface if the ray span defined by α and β is transverse, that is, $F(\alpha) \leq \tau < F(\beta)$ or $F(\beta) \leq \tau < F(\alpha)$. The actual intersection point γ on the surface is calculated as

$$\gamma = \alpha + (\beta - \alpha)\left(\frac{F(\gamma) - F(\alpha)}{F(\beta) - F(\alpha)}\right) \tag{2}$$

Once the intersection point γ has been found, the surface normal at γ, and hence the intensity, can be computed.

The normal at γ is calculated by trilinearly interpolating the normals of the eight voxels bounding the unit cube. The normal, G at a voxel located at (x, y, z) is calculated using difference approximation, that is

$$\begin{aligned} G &= (g_x, g_y, g_z), \\ g_x &= F(x+1, y, z) - F(x-1, y, z), \\ g_y &= F(x, y+1, z) - F(x, y-1, z), \\ g_z &= F(x, y, z+1) - F(x, y, z-1). \end{aligned} \tag{3}$$

With the normal, a shading technique, such as Phong shading, can be used to calculate the intensity at γ. Direct surface rendering has been used to produce high quality scientific visualisations of CT, MRI and simulation data, examples of which can be seen in figure 2.

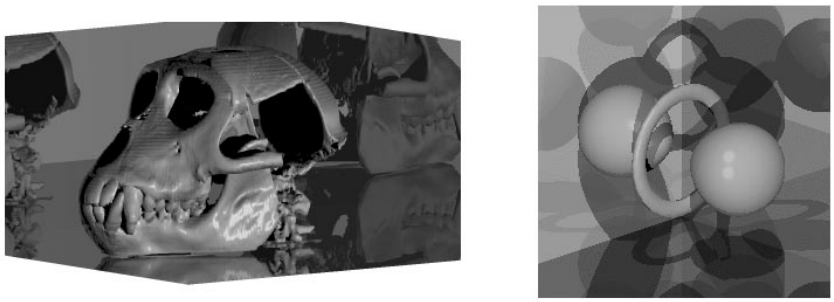

Figure 2: Direct Surface Rendering of a CT monkey head and the AVS hydrogen data set.

Voxelisation of triangular meshes and then direct surface rendering of the volume data has been shown to be faster than the ray tracing of the mesh. The interested reader is directed to the voxelisation work [6] for further details. Volume representation and rendering has been chosen to produce images of volume data calculated from the generated equations because it produces fast and high quality visualisations of the surfaces. The method by which the equations are generated is described in the next section.

3 Genetic Algorithms

It would be impossible to enumerate the set of all surfaces that can be produced by functions $f(x, y, z) = 0$. The space in which these functions exist is so large that it would be impossible to search it in order to find a surface that matches one which a user has in mind. Even if it were possible, it would be difficult for the user to describe the surface in a way such that the difference between two surfaces can be measured. Often users just wish to explore the search space looking for a desirable surface. A random search could be used, but quite often a user may wish to search for variants of an existing shape.

Genetic algorithms [2,16] have been applied in situations where the search space is large, and the method is closely based upon the means by which species propogate. Populations can be regarded as vessels by which DNA is passed from generation to generation. The DNA which produces the fittest organism is that DNA which is most likely to be passed onto the next generation, since its host organism is likely to breed. The fittest members of each population should attract and breed with each other, and their offspring may take the population to new levels of fitness because they have a mixture of the DNA of two successful parents. Members of the population will eventually become ideal, but may miss out on some aspect that could make them even more successful simply because their forbearers did not have it encoded in their DNA. Mutations can provide members with advantages and disadvantages the species may not otherwise possess. If the mutation has a positive effect, that member will be fitter than most, and the DNA has a good chance of being propogated.

In using this analogy, the functions of fitness measurement, and breeding with occasional mutation are those that must be applied to problems if this method for solution is to be used. Genetic algorithms have been applied within computer graphics by many researchers – most notably Sims for the breeding of images [17] and physically realistic motion [18].

An initial population is generated from members of the search space. In this case the population would consist of random functions, or a selection from a set of traditional surfaces such as the torus, sphere and ellipse. The fitness measurement can be user choice – those surfaces that a user prefers from the current population are those which are bred to produce the new population.

Each surface is the isosurface produced from the function when $f = 0$. The function is an equation which is represented as a tree for ease of manipulation. For example the torus of figure 1 is represented by the tree in figure 3.

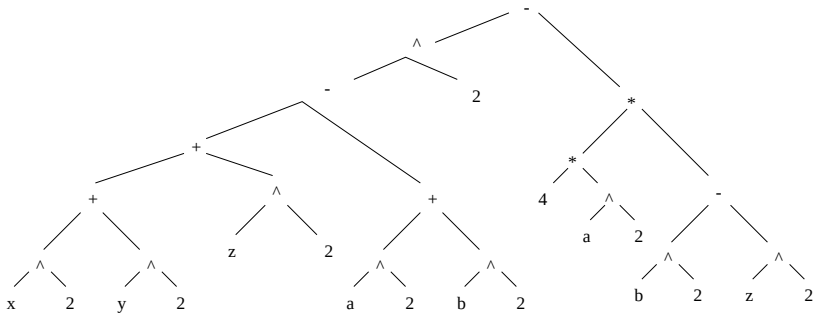

Figure 3: The equation for a torus represented by a binary tree.

The main breeding operation is that of *crossover*. In this process two parents are combined to give two offspring. Using the tree representation for equations, a node is chosen in both trees, and the nodes and their subtrees are swapped between the parents (figure 4). The nature of this operation means that different offspring can be produced by identical parents if the two chosen nodes are different.

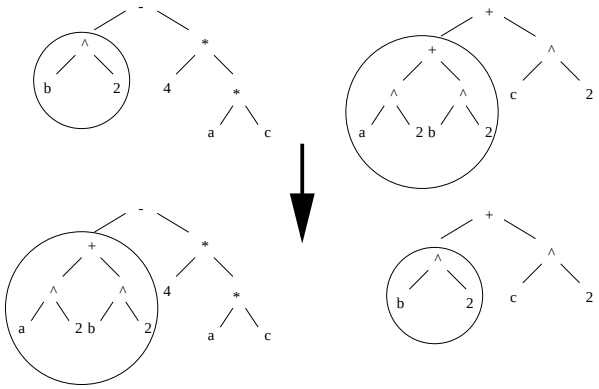

Figure 4: Crossover achieved by swapping two subtrees.

The second genetic operation is that of *mutation*. In this process a node is chosen within one tree, and has its value changed in order to produce a second tree (figure 5).

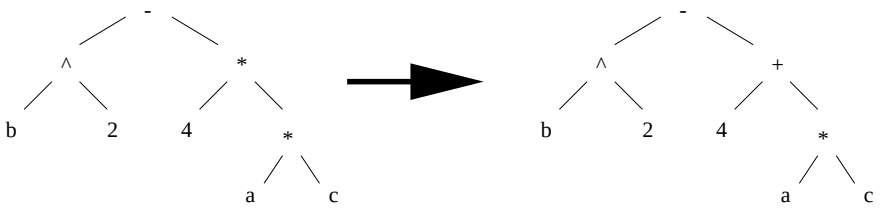

Figure 5: Mutation of one tree into another.

Although the implementation of these operations is relatively straightforward, there are many choices to be made with regards to the types of mutations and crossovers allowed. The main difficulty is the power operator. If the right hand child of any power operator is replaced during crossover by a subtree containing variables, the resulting calculation may overflow. Similar effects may happen if the right hand child mutates to a variable or a large number. It has been found that the following restrictions are suitable and produce varied results. To complete this work, more investigation would be needed into the effects of enabling some of the dissallowed operations.

For mutation a valid node is chosen randomly from the tree. The operators divide, add, multiply and subtract are allowed to mutate into each other with equal probability. They cannot mutate into the power operator. A power operator, and the right hand child of a power operator are considered to be invalid nodes, and are therefore not allowed to mutate. The free variables x, y and z are allowed to mutate to each other. Numbers are allowed to mutate by a random percentage. Figure 6 shows some of the strange and ordinary first generation mutations from the equation for a torus.

For cross-over an *operator* is chosen at random from both trees, and the whole subtree pointed to by that node is swapped with that from the other tree. The root node operator is excluded from this operation as it would result in no genetic information being passed from that parent. Figure 7 shows some results from the cross-over of a torus and the equation for a heart – $(2x^2 + y^2 + z^2 - 1)^3 - (\frac{1}{10})x^2z^3 - y^2z^3 = 0$.

4 Implementation

The work contained within this paper has been implemented using a modular system consisting of a *genetic breeder*, an *equation renderer* and an *equation selector*.

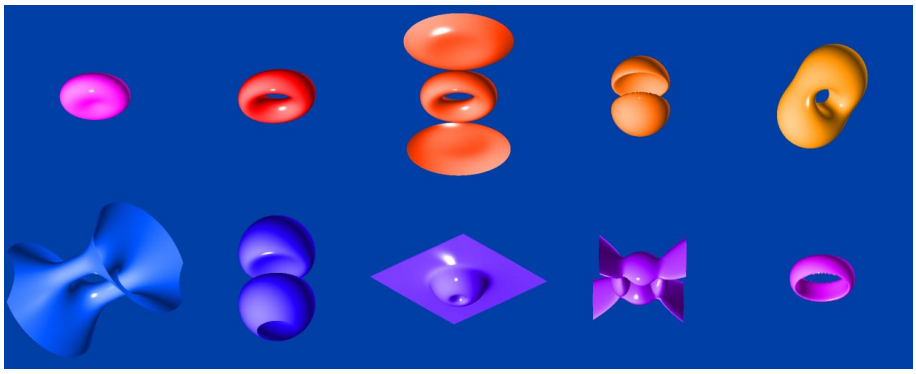

Figure 6: Mutations from a torus.

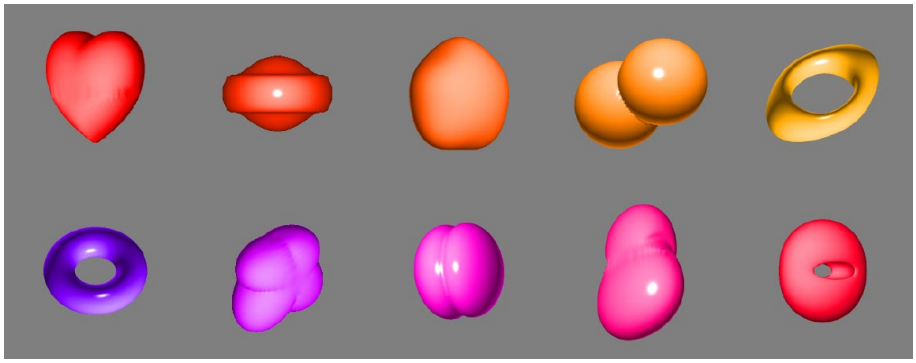

Figure 7: Surfaces (right) produced from the crossover of a torus and heart (left).

- *Genetic Breeder*: This module accepts as input a number of equations and a breeding description and produces as output a new list of equations which will be the next generation. Example input would be:

```
2                          # Number of equations
4                          # Constants in first equation
3                          # Value for constant 1 (a1)
2                          # Value for constant 2 (a2)
0.9                        # a3
1                          # a4
xx*a0*yy*a1*+zz*a2*-a3+    # Postfix notation for equation
3                          # Constants in second equation
12                         # a5
5                          # a6
4                          # a7
xx*yy*+zz*+a4a4*a5a5*+-xx*yy*+zz*+a4a4*a5a5*+-*a6a4*a4*a5a5
*zz*-*-                    # Torus (postfix notation)
2                          # Equations in next generation
m2                         # Generate a mutation of eq. 2
c1,2                       # Cross-over with parents 1 and 2
```

- *Equation Renderer*: This module accepts as input a number of equations and renders them to an output image. The rendering process is carried out by voxelising each equation and using direct surface rendering. Example input would be:

```
60 60 60                   # Size of voxel data set
```

```
10 10                       # Equations to be displayed 10x10
3                           # Constants in first equation
12                          # a1
5                           # a2
4                           # a3
xx*yy*+zz*+a1a1*a2a2*+-xx*yy*+zz*+a1a1*a2a2*+-*a3a1*a1*a2a2
*zz*-*-                     # Torus
...
```

- *Equation Selector*: This module accepts as input an image and allows the user to make choices as to the equations to be used for breeding. The output will be the chosen equations and a breeder description which will be based on the order of relevance / preference.

A breeder description was deemed important in order for the process to have as much flexibility as possible. For example, the first surface the user chooses is deemed to be the most important, and so will be used as the parent of many offspring. The lower down the selection list / ranking an equation is, the less chance it will have of being used as a parent. The breeder description will indicate the mutations and cross-overs to occur for the current generation, and can be generated algorithmically, with a few parameters controlling mutation rate, etc. The flexibility is provided by the fact that the parameters could be tweaked, although this has not yet been considered.

Some results have already been presented in this section, and more detailed results along with some timing information will be given in the final section.

5 Results and Conclusions

The images within this paper have been produced by voxelising the surface equations, and then using the direct surface rendering technique to ray trace them. It takes around 10 seconds on a 233MHz Pentium II processor to voxelise and render the most complex equations within this paper. The objects in figures 6 and 7 have been voxelised on a grid of $60 \times 60 \times 60$ at an image size of 500×500. It has been found that 200×200 images voxelised on a $20 \times 20 \times 20$ grid are adequate and take just a couple of seconds to generate. Another advantage of using the direct surface rendering technique is the fact that the visualisation of the implicit surfaces is simplified – it is no longer necessary to employ numerical techniques or complicated rasterisation algorithms [5].

With regards to the modular system, the *genetic breeder* produces its results fairly instantly. The *equation selector* relies on human interaction, and as a result the run-time is dependent upon the amount of time the user takes to select the interesting surfaces. The main bottleneck in the system is the *equation renderer*. At present about 100 surfaces are presented to the user at each generation. This maybe considered to be a high number, but it was found that a population of this size will give ample choice to the user for equations to breed for the next generation. The rendering stage could easily take place in parallel and greatly reduce the time taken to render so many surfaces.

In considering the success of this method, the motivation behind the work should be examined. The main aim was to allow users to define and explore surfaces with ease. The modular system outlined in this paper allows just that. As can be seen from the results in this paper, many fascinating surfaces can be discovered with very little human interaction. Selecting surfaces to produce new similar surfaces is an intuitive action. Mostly the new

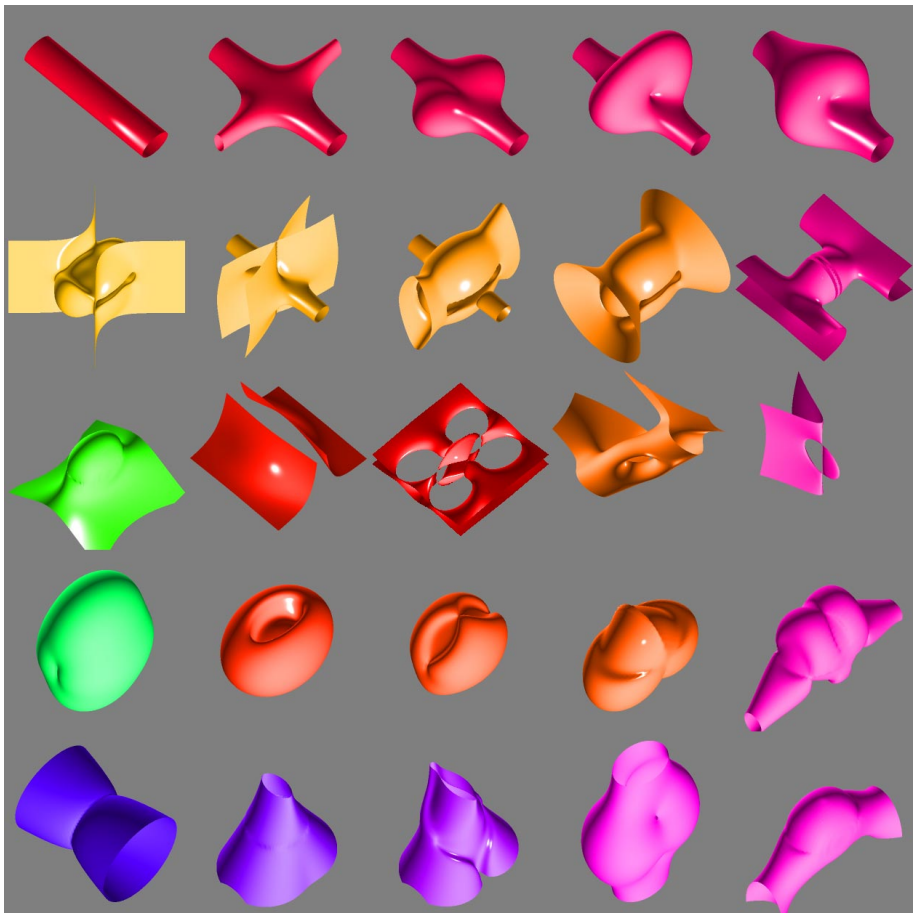

Figure 8: Genetically bred surfaces.

generation will have identifiable characteristics from the previous generation of surface, but there will be some surfaces which seem to bear no relation. Quite often these surfaces can be very useful, and can direct the search into new areas. Many of the interesting surfaces in figure 8 have been produced by playing around by selecting the most exciting surfaces in each generation.

This genetic based approach is much more effective than generating equations randomly. By breeding two surfaces together many times a set of similar shapes can be produced (eg. the top line of figure 8). Randomly generated surfaces do not often produce ”usable objects”, and the user would soon grow tired of such an exhaustive search. The natural interface and the fact that many of the surfaces in one generation contain easily recognisable characteristics from the previous generation means that the user has more control (and feels that control) over the whole surface generation process.

As yet it is difficult to have a surface in mind, and set out specifically to create that surface. Perhaps one reason for this is that breeding two surfaces together (via cross-over) in order to create a surface which bears a strong resemblance to the blend of the two surfaces will not usually produce the desired result due to the nature of the process. It could be that allowing these sorts of operations – for example blending implicit surfaces, could aid the process, and is something that remains for future work.

The results of this work are significantly different to that of the shape synthesiser [1]. Glassner created a system which allows the user to transform surfaces into new surfaces – enabling the user to have direct control over the process. This system allows users to generate new surfaces without being limited to changing existing shapes. This can often lead to unusual

and spectacular surfaces.

References

[1] A. S. Glassner. A shape synthesizer. *IEEE Computer Graphics and Applications*, 17(3):40–51, May-June 1997.

[2] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[3] C. Thornborrow and A. Hobden. Genetic programming for easy 3D texture generation. In *Proceedings of 13th Annual Conference of Eurographics (UK Chapter) (Loughborough, March 28-30, 1995)*, pages 107–116, March 1995.

[4] M. W. Jones. An efficient shadow detection algorithm and the direct surface rendering volume visualisation model. In *Proceedings of 15th Annual Conference of Eurographics (UK Chapter) Norwich*, pages 237–244, March 1997.

[5] G. Taubin. Rasterizing algebraic curves and surfaces. *IEEE Computer Graphics and Applications*, 14(2):14–23, March 1994.

[6] M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum*, 15(5):311–318, December 1996.

[7] A. Kaufman. An algorithm for 3D scan-conversion of polygons. *Proc. of Eurographics '87, Amsterdam, The Netherlands*, pages 197–208, August 1987.

[8] S. W. Wang and A. E. Kaufman. Volume sampled voxelization of geometric primitives. In *Proc. Visualization 93*, pages 78–84. IEEE CS Press, Los Alamitos, Calif., 1993.

[9] T. A. Galyean and J. F. Hughes. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991)*, volume 25(4), pages 267–274. ACM SIGGRAPH, New York, July 1991.

[10] T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.

[11] P. Sabella. A rendering algorithm for visualizing 3D scalar fields. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 51–57. ACM SIGGRAPH, New York, August 1988.

[12] L. Carpenter R. A. Drebin and P. Hanrahan. Volume rendering. In *Proc. SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988)*, volume 22(4), pages 65–74. ACM SIGGRAPH, New York, August 1988.

[13] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[14] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2:227–234, 1986.

[15] W. E. Lorensen and H. E. Cline. Marching Cubes : A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87 (Anaheim, Calif., July 27-31, 1987)*, volume 21(4), pages 163–169. ACM SIGGRAPH, New York, July 1987.

[16] M. Gen and R Cheng. *Genetic Algorithms and Engineering Design.* John Wiley and Sons, 1997.

[17] K. Sims. Artificial evolution for computer graphics. In *Proc. SIGGRAPH '91 (Las Vegas, Nevada, July 28-August 2, 1991)*, volume 25(4), pages 319–328. ACM SIGGRAPH, New York, July 1991.

[18] K. Sims. Evolving virtual creatures. In *Proc. SIGGRAPH '94 (Orlando, Florida, July 24-July 29, 1994)*, volume 28(2), pages 15–24. ACM SIGGRAPH, New York, July 1994.