Turk J Elec Engin, VOL.12, NO.1 2004, © TÜBİTAK

An Implicit Surface Modeling Technique Based on a Modular Neural Network Architecture

Manuel CARCENAC
Computer Engineering Department, Eastern Mediterranean University,
Gazimagusa, via Mersin 10-TURKEY
e-mail: manuel.carcenac@emu.edu.tr

Abstract

Independently from artificial intelligence applications, an artificial neural network can be viewed as a powerful tool for function reconstruction. Previous papers used this property to model an implicit surface out of some control points by reconstructing its underlying scalar field. Such an approach requests the neural network to memorize the control points, which has turned problematic for complex surfaces. In our paper, we show that this problem can be efficiently tackled by adapting the architecture of the neural network to the features compounding the surface: by learning first these features independently and then blending them gradually together, our modular architecture readily comprehends the whole surface. As an example, we model the surface of an animated human body. This approach could eventually help model 3-D textures and be used as well for more classic applications of neural networks.

Key Words: *Geometric Modeling, Implicit Surfaces, Neural Networks, Modular Architecture, Levenberg-Marquardt, Ray Casting.*

1. Introduction

Artificial neural networks have proven to be a versatile tool in artificial intelligence due to their ability to generalize some partial knowledge. In [1,2], we showed that this property can be used to model an implicit surface that is required to pass by some control points: once a neural network has memorized these control points, which represent only a small part of the surface, it will generalize this scattered data and thereby reconstruct the entire surface. We also constrain the surface to comply with some given normals at the control points. Until now, the main shortcoming of such an approach for the network has been learning the control points that pertain to some complex surfaces.

Our paper addresses this issue by presenting a generic and modular neural network architecture for the successive memorization of different features of the surface. This results in a sequence of basic and swift memorization steps, with each surface feature taught individually by a given module. Then, the features are blended together by merging the modules two by two. An original and general mechanism implements this operation: the last hidden layers of the two modules are taken as input to another module that is trained to blend their associated features. Each module, whether in charge of memorizing a single feature or of blending several features, is trained separately. Thus, the total number of weights and biases to train simultaneously remains limited, allowing the use of the highly efficient 2^{nd} order Levenberg-Marquardt method.

The plan of the paper is as follows: first, in Section 2, we present the principle of the approach and the main issues related to the memorization of the control points. Section 3 details our modular architecture and how it gradually learns the surface. Due to the generality of this architecture, we had to develop some recurrent schemes, explained in Section 4, in order to express the network output and its derivatives. Section 5 pertains to the specifics of the learning procedure and Section 6 to the rendering of the surface. In Section 7, we present the surface modeling of an animated human body that was carried out with our approach. Finally, Section 8 lays out the future work we are currently planning and Section 9 compares our method with related approaches.

2. Modeling an Implicit Surface With a Neural Network

As a classical tool in artificial intelligence, artificial neural networks proceed by first memorizing a limited amount of data regarding a certain problem. Then, they adapt to any new data by expanding their previously memorized knowledge. For example, if we consider the general problem of pattern recognition, the network is taught the characteristics of certain objects and the classes to which each of these objects belong. When presented with a completely new object, it interpolates between the class memberships of the memorized objects in order to obtain a plausible class membership.

From a general mathematical standpoint, a neural network is a powerful interpolation tool that reconstructs a function from a limited set of scattered points over which the value of the function is provided. Actually, implicit surface modeling may be viewed as a problem of so-called function reconstruction [3-5]. Let us assume the implicit surface is based on a 3-D scalar field $\Phi(x_0, x_1, x_2)$ and is made up of the points at which this field is null. Then, the objective is to build a suitable field out of a few control points on the surface where we have the local constraint $\Phi = 0$. The starting point of our approach is to use a neural network to interpolate the field between these control points. Also, we impose the field gradient over the control points in order to control the local orientation of the surface and have non null field values inside and outside it. The data to be provided for a given control point are its coordinates x_k and the coordinates n_k of the outgoing unit normal. The constraints at this point are then $\Phi = 0$ and $\partial\Phi/\partial x_k = -\,s\; n_k$, with s slope of the field at the surface.

In practice, modeling an implicit surface with a neural network can be split in two phases. The first is the learning phase, when suitable weights and biases for the connections are found so that the output field complies with the control points. Then, once proper connections are settled, the rendering phase consists of elaborating a 3-D image of the surface, requiring the calculation of Φ over a number of points.

There are several serious issues related to the learning phase. First, the control points only locally constrain the field, and we must ensure that the interpolation between them yields a surface of good quality, particularly regarding its smoothness. Secondly, this phase is implemented as an iterative punctual search in the space of weight and bias variables aimed at minimizing the total error over the control points. If we try to learn globally a complex surface, this local search will likely end up trapped in a local minimum from which it cannot escape, or in a flat trough from which it will take far too long to exit. Indeed, many features of the surface will impose contradictory search directions that should result in a global stalemate.

3. A Modular Architecture for Localized and Gradual Learning

We initially relied upon a standard multilayer feedforward neural network. A regular succession of large layers of neurons; it was a monolithic architecture that implied global learning of the surface, at the risk of

being unable to simultaneously comprehend all its features. Since then, we have developed an architecture that learns the local features of the surface one after the other and blends them progressively together, resulting in a sequence of much easier and faster learning steps.

The surface is first decomposed into elementary features of simple geometry, represented by groups of control points that tend to be localized in space. Each of these groups is learnt readily by a subnetwork, or module, independently from the other groups (steps 1, 2, 4 of Figure 1). Then, couples of such basic modules are merged together at their ends to form a new module that easily models the blending of their two elementary features (step 3 of Figure 1). Through continued merging of module pairs, we finally reconstitute the entire surface. The resulting architecture has a binary tree structure, with the "leaf modules" learning the elementary features and the "node modules" enforcing the blending of both features and groups of previously blended features (step 5 of Figure 1).

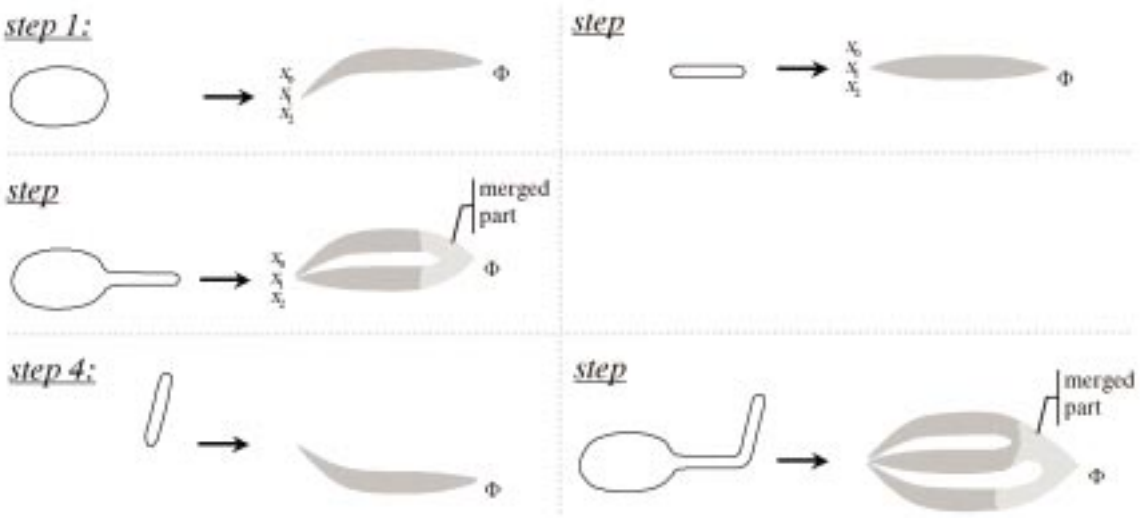

Figure 1. Modular architecture for 3 blended features.

We now present the specifics of this modular architecture. Each of the modules is a set of small layers of neurons, typically less than 10, with forward connections between them. A forward connection between two layers implies that all the neurons of the first layer connect to each neuron of the second layer (it will be represented globally in the following figures by a single arrow between the layers). Apart from the input layer that contains 3 neurons (for x_0, x_1, x_2) and the output layer that has only 1 neuron (for Φ), we assume a limited and constant number of neurons N for all other layers. In theory, the mapping of an arbitrary function requires a network with only one "hidden" layer of neurons [6,7], found between the input and output layers i and o. However, because 2 hidden layers are known to facilitate faster learning [8], we choose for a leaf module the basic multilayer feedforward neural network architecture with 2 hidden layers a and b of size N (Figure 2).

Figure 2. Leaf module.

All three connections $i \rightarrow a$, $a \rightarrow b$ and $b \rightarrow o$ must be taught simultaneously to memorize the single feature related to the module.

The node modules are built by combining and reconfiguring the last connections $b_1 \rightarrow o_1$ and $b_2 \rightarrow o_2$ of two previously built modules. The connections from i until b_1 and b_2 form the underlying memory inherited from previous learning steps and remain unchanged, while the $b_1 \rightarrow o_1$ and $b_2 \rightarrow o_2$ connections that were used for independent teaching of the two modules must now be discarded to allow blending. The blended part of the new module is itself a neural network that maps the function of inputs b_1, b_2 and new output o', and underlies the blending of the previous surfaces. Similarly for the leaf modules, we assign to this neural network two hidden layers a', b' (also of size N) as shown in Figure 3.

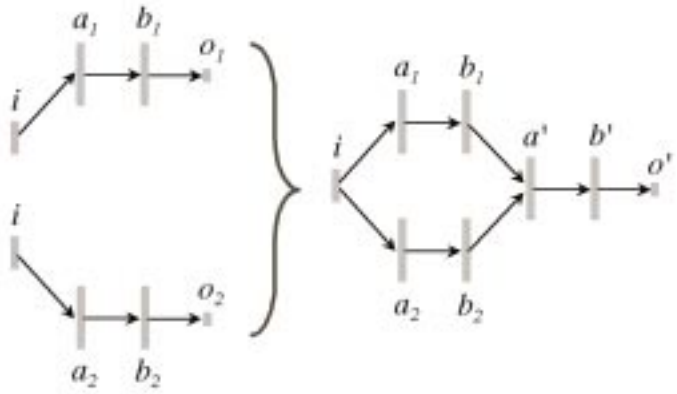

Figure 3. 2 leaf modules merged into a node module.

Only the connections $b_1 \rightarrow a'$, $b_2 \rightarrow a'$, $a' \rightarrow b'$ and $b' \rightarrow o'$ of the node module are then taught to comply with all the features of the leaf modules that the node encompasses. In the same manner, the merging of the above node module with a third leaf module (shown in Figure 4) is done by teaching the new connections $b' \rightarrow a''$, $b_3 \rightarrow a''$, $a'' \rightarrow b''$ and $b'' \rightarrow o''$ to comply with all the features of the three leaf modules.

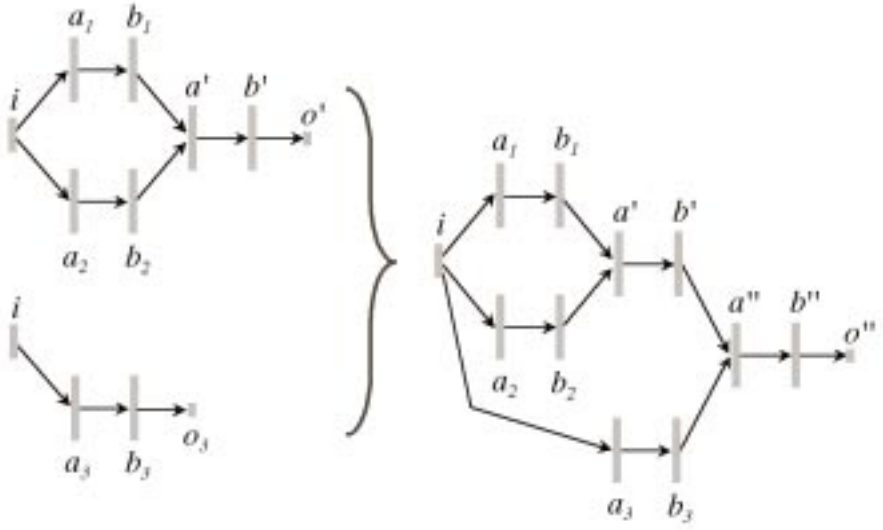

Figure 4. Node module merged with a third leaf module.

Note that since N is small (≤ 10), each learning step, whether it is for a leaf or particularly for a node module, involves relatively few weight and bias variables, far less anyway than with a monolithic architecture. It therefore allows the use of a 2^{nd} order learning method that is very efficient but remains practical only for a limited number of variables (see Section 5).

4. A Recurrent Schemes for the Output of a Module

The architecture layed out in the previous section is very general and can be instantiated into a multitude of forms. For this reason, we have developed some recurrent and versatile schemes to express the output Φ of the network, as well as its gradient, in function of the inputs x_0, x_1, x_2. A given module is described by its input and output layers *inl* and *outl* and by some general information about any one of its layers l, regarding its connectivity in particular:

- $h \in D\ (l)$ means h is a direct predecessor layer of l (direct forward connection $h \rightarrow l$),
- $h \in C\ (l)$ means h is a predecessor layer of l connecting directly or indirectly to it, and
- $i \in L\ (l)$ is the index (first value 0) of a neuron of layer l.

We choose a standard activation function [6] whose range $(\ -1.7159\ ,\ +1.7159\ )$ is centered on 0: $a\ (H\ ) = 1.7159\ \times\ \tanh\ (2\,H\,/\,3\ )$. The value O_i^l for neuron i of layer l is then computed as follows: if l is the input layer, it is directly one of the input coordinates, or else it is the result of the activation function applied over the net input H_i^l of this neuron:

$$O_i^l = \left[\begin{array}{l} if\ l = inl : x_i \\ \\ else : a\left(H_i^l\right) \end{array} \right. ;\ \text{in particular: } \Phi = O_0^{outl}$$

The net input is the weighted and biased sum of the values of the neurons that connect directly to the given neuron: $H_i^l = b_i^l + \sum\limits_{l' \in D(l)} \sum\limits_{i' \in L(l')} w_{i\,i'}^{l\,l'}\ O_{i'}^{l'}$

The gradient of Φ is obtained by analytical derivation of the above recurrent scheme:

$$\frac{\partial \Phi}{\partial x_k} = {}^kG_0^{outl} \text{ with } {}^kG_i^l = \frac{\partial O_i^l}{\partial x_k} = \left[\begin{array}{l} if\ l = inl : \delta_{ik} \\ \\ else : a'\left(H_i^l\right)\ {}^kI_i^l \end{array} \right. ;\ {}^kI_i^l = \frac{\partial H_i^l}{\partial x_k} = \sum_{l' \in D(l)} \sum_{i' \in L(l')} w_{i\,i'}^{l\,l'}\ {}^kG_{i'}^{l'}$$

(δ_{ik} is Kronecker's symbol).

The learning method also requests the partial derivatives of Φ and $\nabla\Phi$ relative to any weight or bias $w_{j\,j'}^{h\,h'}$ or b_j^h. The corresponding recurrent schemes, obtained by direct analytical derivation, replace the commonly used backpropagation scheme [9] (faster, but which does not provide the derivatives of the output gradient). The derivatives of O_i^l and ${}^kG_i^l$ relative to $w_{j\,j'}^{h\,h'}$ are obtained straightforwardly (the formulas relative to b_j^h are similar):

$$\frac{\partial\, O_i^l}{\partial w_{j\,j'}^{h\,h'}} = \left[\begin{array}{l} if\ l = inl : 0 \\ \\ else : a'\left(H_i^l\right)\ \frac{\partial H_i^l}{\partial w_{j\,j'}^{h\,h'}} \end{array} \right. \quad ; \quad \frac{\partial\, {}^kG_i^l}{\partial w_{j\,j'}^{h\,h'}} = \left[\begin{array}{l} if\ l = inl : 0 \\ \\ else : a''\left(H_i^l\right)\ \frac{\partial H_i^l}{\partial w_{j\,j'}^{h\,h'}}\ {}^kI_i^l + a'\left(H_i^l\right)\ \frac{\partial\, {}^kI_i^l}{\partial w_{j\,j'}^{h\,h'}} \end{array} \right.$$

As for the derivatives of H_i^l and ${}^kI_i^l$, we must consider three cases: if $w_{j\,j'}^{h\,h'}$ appears directly in the corresponding expression, if it is contained in one of the terms $O_{i'}^{l'}$ (or ${}^kG_{i'}^{l'}$), or finally if it has no influence at all over H_i^l (or ${}^kI_i^l$).

$$\frac{\partial H_i^l}{\partial w_{j\,j'}^{h\,h'}} = \begin{cases} if\ h = l : \delta_{i\,j}\ O_{j'}^{h'} \\ if\ h \in C\,(l) : \sum\limits_{l' \in D(l)}\ \sum\limits_{i' \in L(l')}\ w_{i\,i'}^{l\,l'}\ \frac{\partial\, O_{i'}^{l'}}{\partial w_{j\,j'}^{h\,h'}} \\ else : 0 \end{cases}$$

$$\frac{\partial\, {}^kI_i^l}{\partial w_{j\,j'}^{h\,h'}} = \begin{cases} if\ h = l : \delta_{i\,j}\, {}^kG_{j'}^{h'} \\ if\ h \in C\,(l) : \sum\limits_{l' \in D(l)}\ \sum\limits_{i' \in L(l')}\ w_{i\,i'}^{l\,l'}\ \frac{\partial\, {}^kG_{i'}^{l'}}{\partial w_{j\,j'}^{h\,h'}} \\ else : 0 \end{cases}$$

5. The Learning Procedure

The learning phase for a given module consists of finding values for all or part of its weights and biases such that it complies with all the control points related to it. At any control point p of the module, the errors over the field $\Delta_p\Phi = 0 - \Phi$ and over its gradient coordinates $\Delta_p\partial\,\Phi/\partial\,x_k = -\,s\,n_k - \partial\,\Phi/\partial\,x_k$ must thus be minimized. Globally, we minimize the overall "standard" error, sum of these squared errors, which encompasses all the control points of the module:

$E_s = \frac{1}{2\,N_s}\sum\limits_p \left((\Delta_p\Phi)^{\,2} + \left(\gamma\ \Delta_p \frac{\partial\,\Phi}{\partial\,x_0}\right)^2 + \left(\gamma\ \Delta_p \frac{\partial\,\Phi}{\partial\,x_1}\right)^2 + \left(\gamma\ \Delta_p \frac{\partial\,\Phi}{\partial\,x_2}\right)^2 \right)$ with N_s being the number of error terms for the module and γ constant aimed at easing the convergence process.

Although an infinity of surfaces may formally comply with the control points, we should try to obtain the surface which is regular and smooth enough, that is, whose global, mean, curvature is minimized. Hence, in addition to the standard error E_s, we must also minimize a complexity term E_c that quantifies this global curvature. Classically, the formula chosen for E_c is but a rough approximation of it. In particular, the so-called weight decay method [6] that we use merely assumes:

$E_c = \frac{\lambda}{2}\ \sum\limits_V\ V^{\,2}$ with variables V (weights and biases) to be taught in the module.

Overall, the criterion to be minimized is $E_{sc} = E_s + E_c$.

As noticed in Section 3, due to our modular architecture, relatively few variables are to be taught at the same time. Thus, instead of the 1^{st} order gradient descent method commonly used to teach neural networks [6,10,11], we can employ the very efficient 2^{nd}order Levenberg-Marquardt method [11] applied to least square minimization. It consists of the 2^{nd} order Newton minimization method with the Hessian of E_s approximated by $\frac{1}{N_s}J^t\,J$. J is the Jacobian of $[\,e_s]$, a vector containing the error terms $\Delta_p\Phi$ and $\gamma\ \Delta_p\partial\,\Phi/\partial\,x_k$ of all the control points related to the module (J is calculated with the recurrent schemes of Section 4). The method also includes a diagonal term $\mu\ I$ added to the Hessian of E_{sc} to ensure that the resulting matrix remains nonsingular. The increments $[\,\Delta\,V\,]$ of the variables are obtained by solving the linear system (using Gauss elimination method):

$$(A \ + \ \mu \ I) \ [\Delta V] = \quad B \text{ with } A \ = \ \nabla^2 E_s \ + \ \nabla^2 E_c \ \approx \ \frac{1}{N_s} J^t J \ + \ \lambda \ I$$

$$\text{and } B \ = \ - \ (\nabla E_s \ + \ \nabla E_c) \ = \ - \ \left(\frac{1}{N_s} J^t [e_s] \ + \ \lambda \, [V] \right)$$

As for the parameter μ, it is dynamically adjusted by the minimization algorithm (by a ratio $\theta > 1$) so that it remains just big enough to avoid singularities and ensure an actual decrease of E_s:

$\mu \leftarrow \mu_0$
while $E_s >$ *threshold* :
- *compute* A, B
- *solve* $(A \ + \ \mu \, I) \ [\Delta V] = B$
- *while* $E_s >$ *previous* E_s :
  - $[\mu \leftarrow \mu \, \theta$; *solve again* $(A \ + \ \mu \, I) \ [\Delta V] = B$
- $[V] \leftarrow [V] + [\Delta V]$
- $\mu \leftarrow \mu / \theta$

At the start of this iterative process, the weights to be taught are initialized at random in the interval $[-w_0 \, , \, +w_0]$, whereas the biases are simply initialized at 0. Actually, in the case of an animation in which the surface evolves gradually, it is suitable to initiate the learning phase for the current state of the surface with the weight and bias values of its previous state (assuming two consecutive surface states are only slightly different). The convergence process is then logically much faster (see Section 7).

Finally, we must notice that the surface parts learnt by the leaf modules may exhibit widely varying scales. This can force the activation function to become saturated (that is, take a value of quasi -1.7159 or +1.7159) within the expressions of the derivatives relative to the weights and biases, resulting in a stalled convergence process. Hence, the coordinates of the control points of a leaf module must be normalized. To take into account the spatial distribution of the swarm of control points, this normalization is done within a rectangular bounding box oriented according to its main directions. Proper scaling is applied over each of these directions so that the normalized coordinates remain within the interval $\left[- \, \frac{H}{2} \, , \, + \, \frac{H}{2} \right]$.

6. Rendering the Implicit Surface

We perform rendering using ray casting. This is done by building an octree that represents the surface [12,13] and performing a recursive search within the octree to localize the intersection between a ray and the surface [14]. The octree is built adaptively by determining whether a given voxel is likely to contain part of the surface. If it may be the case, then we subdivide it into 8 subvoxels and so on until we obtain voxels small enough for the image to be rendered.

To make sure voxels that actually intersect the surface were not mistakenly pruned out, we initially used in [1,2] interval arithmetic as described in [13,15]: the interval of all possible values of Φ over the voxel was calculated and the voxel was discarded only if this interval did not contain value 0. However, the output

interval was often too loose since the interval calculations had to be carried out through all the layers of neurons. Therefore, we were forced to evaluate the octree at prohibitively small scales.

Since we were concerned that a similar problem might eventually occur for other "guaranteed" methods, such as the one based on Lipschitz constants [16], we have opted -at least for the time being- for a punctual iterative method, although there is no guarantee in theory that it always yields a correct answer. The search for a null value of Φ within the voxel is thus carried out iteratively with Newton method (we realized that it was in fact faster to do the search within the smallest sphere that contains the voxel). To prevent such a search from failing due to the complexity of the implicit surface, we assume that this surface is contained within the bounding boxes previously used for the normalization of the leaf modules (see Section 5). These boxes, shown in Figure 5, must actually be slightly inflated to include the joining areas between the features.

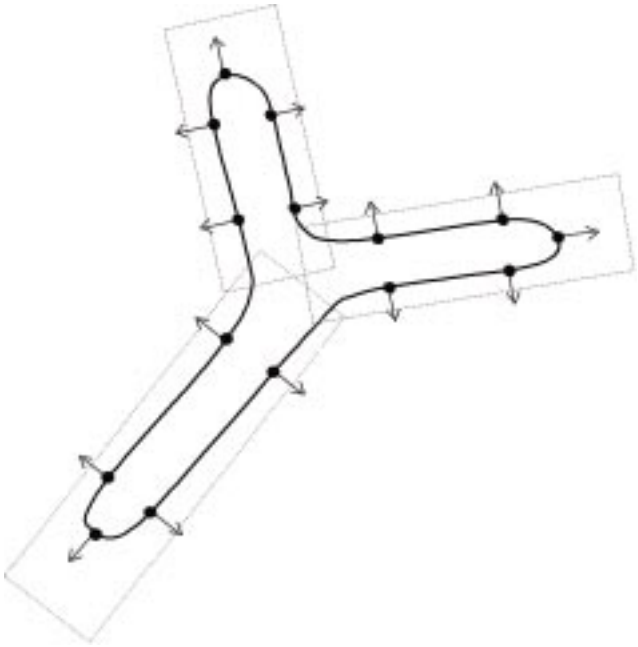

Figure 5. Bounding boxes for rendering.

Then, we systematically divide the main size voxels as long as they intersect one of these bounding volumes. Finally, we obtain some voxels slightly smaller than the bounding volumes they intersect, meaning that these voxels are likely to contain geometrically simple parts of the surface. The iterative search can then be applied, since it is unlikely that it would be thwarted by such simple pieces of surface.

7. Modeling the Surface of an Animated Human Body

In this section, our approach is applied first to the modeling of a human head, and then of a complete human body. To conclude, we will animate this body over a sequence of 50 frames. The following results were obtained using a program written in C language on a 500 MHz Pentium III PC. Anti-aliasing was performed over the images by casting 4 rays per pixel. The values chosen for the parameters are detailed in Table 1.

The human head is subdivided into 3 elementary features each of which is easy to memorize: a bare head void of details, then a nose and finally a mouth. 3 leaf modules are assigned to these features and 2

node modules enforce the blending, first of the bare head with the nose, and then of the resulting subsurface with the mouth. A total of 99 control points and their normals describe the 3 features of the head that is memorized gradually, feature after feature (Figure 6).

Table 1. Parameter values.

s	γ	λ	w_0	H	μ_0	θ	N
5	0.05	1.2×10^{-4}	0.5	2	10^{-4}	2	9

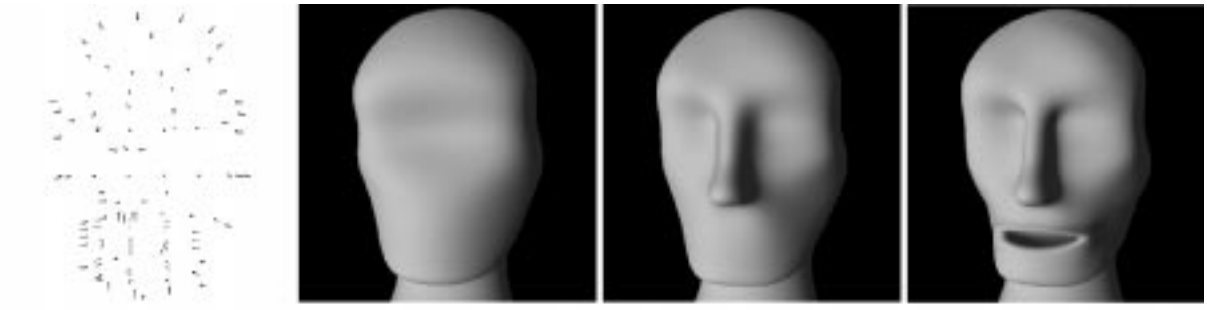

Figure 6. Control points and gradual memorization for the head.

The memorization process in Figure 6 was carried out with an error threshold of 10^{-5}. Table 2 exposes the computing times required to learn and render the head.

Table 2. Computing times for the head.

learning	last view of Fig. 5 - 256×256 pixels
69 s	229 s

As for the complete human body, it includes this same head and is described by 316 control points and their normals grouped into 17 features. These features are learned separately and then gradually blended two by two. The control points and the complete body are shown in Figure 7.

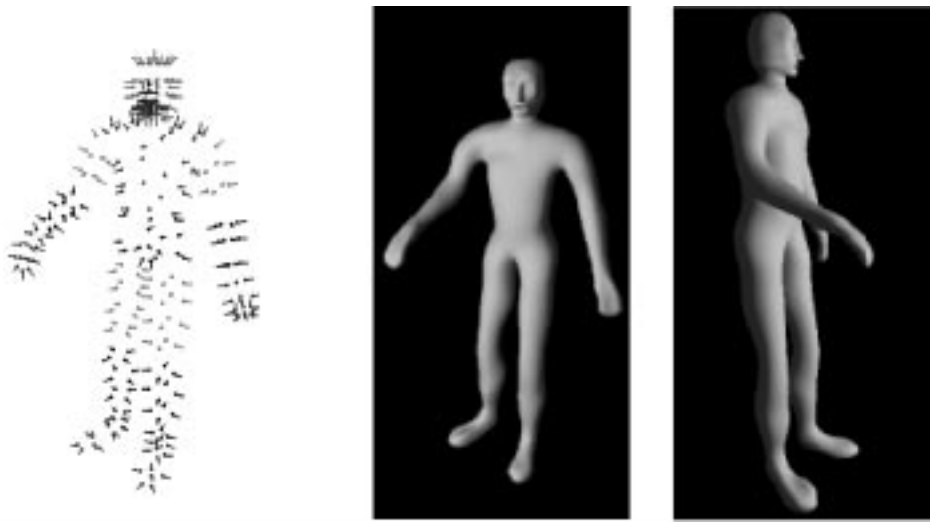

Figure 7. Control points for the complete body; 2 views of the body.

The features and intermediate subsurfaces are blended two by two according to the tree of Figure 8. A few other tested combinations did not seem to substantially change the surface. Apart from corresponding to a most intuitive blending, this tree is organized so as to be approximately balanced. In this way we limit the number of node modules that encompass many control points and that take therefore more time to teach.

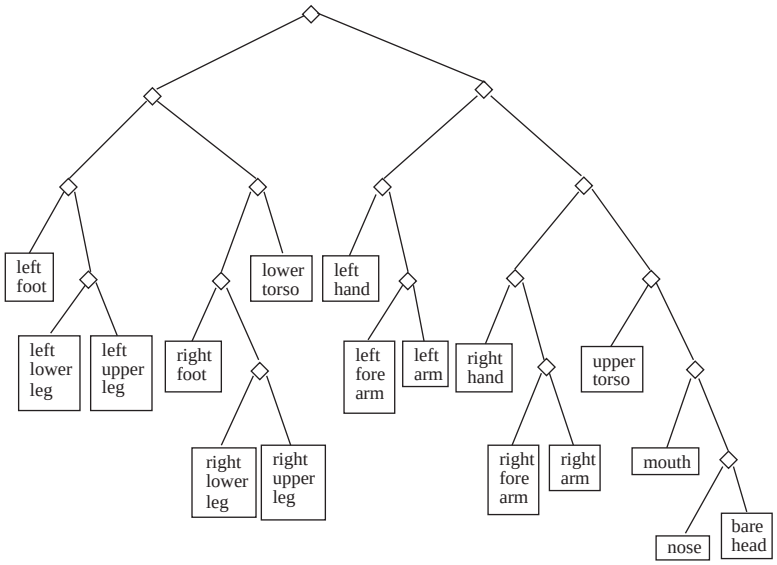

Figure 8. Human body features blended together.

Overall, we teach 17 leaf modules to comply with their associated features and 16 node modules to enforce the blendings. A priori, the error threshold is 10^{-5} but, as an optimization, we slightly relax it for the root node module (1.7×10^{-5}) and its two subnode modules (1.5×10^{-5}) (it does not entail any noticeable change in the surface). The number of iterations and the time required to teach each of the modules are displayed in Figure 9.

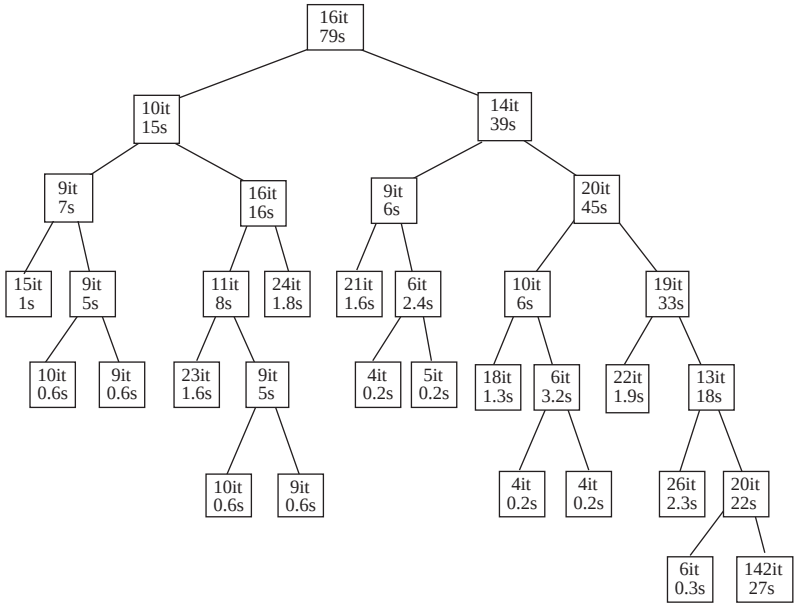

Figure 9. Number of iterations and time spent per module.

It is most interesting to note that the final node modules learn to blend complex subsurfaces easily -that is with a limited number of iterations- although they have just as many connections that can be taught as other node modules whose learning tasks seem a priori much lighter. This result can be interpreted as a justification a posteriori of our modular architecture. Also, the algorithmic complexity of each iteration of these final modules tends to grow linearly with the number of control points. In particular, the costs of the evaluation of J and of the product $J^t J$ grow strictly linearly with this number, but it does not intervene in the cost of the resolution of the linear system.

Some imperfections can be seen in Figure 7, notably in the right foot. They may be related to insufficiencies in the convergence process (see Section 8). Table 3 presents the computing time required to learn the whole body (total time spent over all the modules). It also displays the times required to render the two views of Figure 7.

Table 3. Computing times for the body.

learning	1^{st} view of Fig. 6 128×256 pix.	2^{nd} view of Fig. 6 128×256 pix.
352 s	238 s	272 s

It should be noted that it would be unrealistic to teach this complex surface directly to a monolithic multilayer feedforward neural network with the Levenberg-Marquardt method. That would indeed necessitate far more variables (weights and biases) than it is possible in practice to handle with this numerical method. However, an early attempt to model a similar body with such a network and the conjugate gradient method (with line minimization) [9,11] was reported in [1]. Unfortunately, the result was quite dispiriting: it took up to 763 hours (at 350 MHz) to obtain a very rough approximation of the surface.

We now present a small, proof-of-concept, animation of the body. Starting with the previous configuration, we gradually raise and flex the body's two arms and its right leg over a sequence of 50 frames. As expected, the incremental learning of each new state of the surface is then much faster than for the initial state (see Table 4).

Table 4. Computing times for the animation.

incremental learning of a new state	images from animation 128×256 pix.
average: 38 s minimum: 15 s maximum: 62 s	average: 235 s

A few frames of the animation are shown in Figure 10.

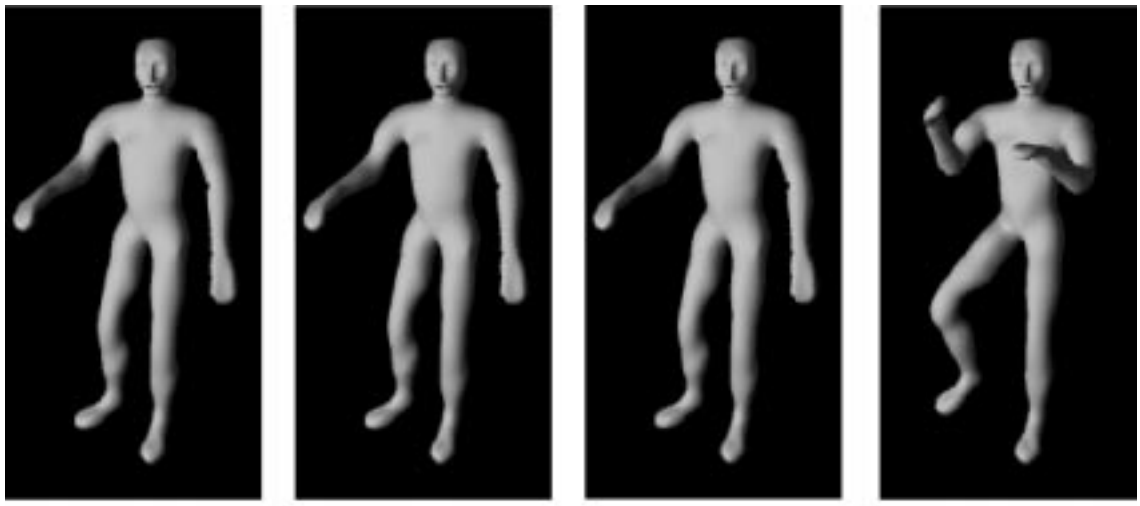

Figure 10. A few frames from the body animation.

We notice that the incremental learning allows on average close to a tenfold decrease in the computing time, as compared to the 352 s required for the initial state.

8. Future Development

The main problem encountered so far by our approach is that the convergence process is not robust enough: varying the control points and the values of μ_0 and θ parameters may eventually induce some surface defaults and also some blobs in its vicinity. We believe that the main reason for this lies in the widely different scalings of the variations of the weights and biases during the iterations. A trust-region method [17-19] would be adapted, rather than the Levenberg-Marquardt method, to take into account this phenomenon (since in particular its trust region may be adequately scaled).

The smoothing technique might influence the quality and robustness of the convergence as well: the weight decay method is indeed pretty crude in that it is not formally connected to the global curvature. Furthermore, this technique considers the weights and biases independently from one another and treats their individual positions and influences identically within the neural network. Ultimately, the convergence process may be hampered and the surface obtained may not be the smoothest.

Concerning the actual modeling of the surface, its decomposition into individual features is so far done manually. This does not really represent a problem since the human operator tends to do it naturally and in a most intuitive way. However, for more general purposes, one can envision that such a decomposition might be carried out automatically, likely by some sort of space partitioning.

Eventually, we are looking forward to dynamic recurrent neural networks in order to generalize further our generic architecture. In particular, introducing some feedback connections within some of the modules would allow them to "iterate" a previously memorized surface feature by duplicating it and scaling it down a number of times. This might eventually turn out to be a fruitful generator of 3-D fractal textures.

Incidentally, we have recently obtained some very promising initial results by applying our modular architecture to a pattern recognition application. In this application, the image is partitioned recursively into small patches of 4×4 pixels that are learned independently by the leaf modules. Then, the node modules gradually merge each partial information together. The most obvious advantage is a swift memorization of image patterns made up of a substantial number of pixels, up to 64×64 in our initial tests.

9. Related Work

Our modular architecture may be compared to that of a committee machine [6] which is also modular: each module, or "expert", produces an output that is directly merged with the outputs of the other experts. However, what we merge, and retrain, are the last connections of our modules. We believe that this provides a smoother blending of the surface features.

Two prominent types of neural networks are multilayer feedforward networks, on which the approach developed in [1,2] and this paper is based, and radial basis function (RBF) networks that are used in [5] to model an implicit surface out of control points. This kind of network consists of a single layer of n neurons (with a specific RBF for each of them), n being also the number of control points. Learning then consists of the resolution, with a direct method, of a linear system of size n. However, that becomes impractical for large n, that is for too many control points. To overcome this limitation, a generalized RBF network [6] could be used; it would then be possible to have substantially less neurons than control points. However, as with our approach, that would imply an iterative resolution and the risk of being trapped in a local minimum (a risk that our specific architecture was designed to overcome).

In [5], the field is shaped into a positive area (inside the surface) and a negative area (outside it) by adding close to each control point another control point where the field is non null. As for us, by directly imposing the normal at each control point, we not only shape the field but also provide an accurate control over the local orientation of the surface.

Another approach for elaborating a surface out of some imposed points is that of [3,4]: it consists of finding a suitable set of tangent planes that cover the whole surface. The signed distance function for these planes is then chosen as the 3-D field of the implicit surface. Finally, the surface is triangulated. Actually, this method requires a ρ-dense distribution of points that may imply several thousand points or more. By contrast, our approach requests only a limited number of points that may then be irregularly distributed.

In [20], another approach based on RBF's is presented that allows a far greater number of data points than [5]. Instead of solving a linear system with a direct method, it relies on the so-called Fast Multipole Method to fit the RBF to the data points.

In fact, the objectives of these methods differ significantly from ours. They aim at surface reconstruction out of a substantial set of sample points that are yielded by a sampling device like a 3-D laser scanner. Instead, our method is meant for surface modeling using as few control points as possible (so that they may be readily provided and modified by a human operator). In order to illustrate this difference, we have modeled a torus both with a single module of our architecture and with an RBF neural network trained as in [5]. Our choice for the RBF was the multiquadric $\varphi\ (x,y,z) = \sqrt{x^2+y^2+z^2+c^2}$ with $c^2 = 10^{-3}$ (we found that the term c^2 was necessary to avoid certain artefacts very close to the control points). The field is then given by $\Phi\ (x,y,z) = \sum_{i=1}^{n} \lambda_i \cdot \varphi\ (x-P_i^x, y-P_i^y, z-P_i^z)\ + p + p_x \cdot x + p_y \cdot y + p_z \cdot z$ with n control points P_i and parameters λ_i and p, p_x, p_y, p_z (obtained by solving a linear system). Figure 11 shows, for 48, 96, 160, 336 and 576 control points and their normals (1^{st} row), the surfaces produced with our network (2^{nd} row) and with the RBF network (3^{rd} row).

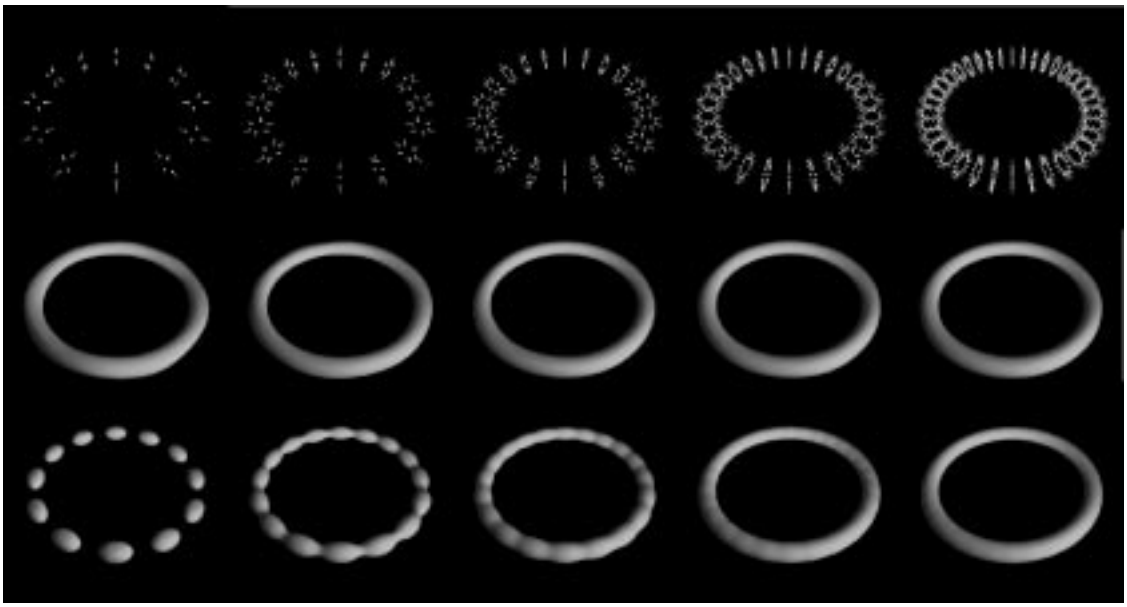

Figure 11. Surfaces obtained for 48, 96, 160, 336 and 576 control points with our network (2^{nd} row) and with the RBF network (3^{rd} row).

Table 5 gives the associated training times, as measured on a 1.7 GHz Pentium IV PC, and numbers of iterations.

Table 5. Computing times and numbers of iterations for the comparison between our network and the RBF network.

number of control points	48	96	160	336	576
number of iterations for our network	73 it.	61 it.	67 it.	58 it.	59 it.
learning time for our network	2.5 s	3.6 s	6.2 s	10.9 s	17.8 s
learning time for the RBF network	1.8 ms	27 ms	0.18 s	1.88 s	8.3 s

It appears from Figure 11 that a multilayer feedforward neural network requires substantially fewer control points to produce a satisfactory surface than a single layer RBF network. In the case of the RBF network, as many as 336 control points still leave the surface riddled with undesirable artefacts (undulations indicating the presence of the control points). Up to 576 control points are necessary to obtain an acceptable surface. Table 5 shows that our approach requires a longer training time, at least for limited numbers of control points. Indeed, it takes many iterations to match our highly non linear multilayer network to the surface of the torus. In contrast, the RBF single layer network requires only one linear system resolution. However, this gap between the two methods diminishes as the number of control points increases: the cost of an iteration of the Levenberg-Marquardt method tends to grow linearly with this number (because of the product J^tJ), whereas the cost of the resolution of the linear system of the RBF network tends to grow with the cube of it.

We finally report the work in [21] that models an implicit surface through the control of a distribution of particles. This approach is quite general in that it does not impose a priori a particular mathematical definition for the scalar field. Actually, it might even be used in conjunction with neural networks.

10. Conclusion

Our method of using a neural network to model an implicit surface seems quite appealing in that it only requires the specification of relatively few control points by the human operator and a very intuitive subdivision of the surface into basic features. The main problem was the memorization of the control points, which is apparently intractable for a complex surface and with a monolithic neural network. However, the modular architecture that we have detailed seems to scale up readily to a substantial number of surface features. It can thereby adapt to quite complex surfaces. Besides, the approach seems well suited to an animation in which an implicit surface gradually evolves over time. One might even envision interactive modeling based on incremental learning of the neural network.

Although we think the architectural part is relatively settled, it is clear that the numerical resolution and -eventually- the smoothing technique must still be improved. Overall, the strategy of adapting a generic and elaborate neural network architecture to the complexity of a surface is still a very open research subject (most particularly with regard to 3-D textures).

Looking toward the future, we also intend to investigate the interest of our approach for more classical application fields of neural networks, pattern recognition in particular.

References

[1] M. Carcenac, A. Acan, "Modeling an Isosurface with a Neural Network," Pacific Graphics 2000 Conference, pp. 165-174, Hong Kong, 2000.

[2] M. Carcenac, "Adapting the Architecture of a Neural Network to the Modeling of an Isosurface", TAINN 2001 Conference, pp. 42-51, Gazimagusa, 2001.

[3] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, "Surface Reconstruction from Unorganized Points," SIGGRAPH 92 Proceedings, Vol. 26(2), pp. 71-78, 1992.

[4] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, W. Stuetzle, "Piecewise Smooth Surface Reconstruction," SIGGRAPH 94 Proceedings, pp. 295-302, 1994.

[5] G. Turk, J.F. O'Brien, "Shape Transformation Using Variational Implicit Functions", SIGGRAPH 99 Proceedings, pp. 335-342, 1999.

[6] S. Haykin, Neural Networks - A Comprehensive Foundation, Second Edition, Prentice Hall, 1999.

[7] R. Hecht-Nielsen, "Kolmogorov's Mapping Neural Network Existence Theorem," Proceedings of the First IEEE International Conference on Neural Networks, San Diego, Vol. 3, pp. 112-114, 1987.

[8] V. Kurkova, "Kolmogorov's Theorem is relevant," Neural Computation, Vol. 3, pp. 617-622, 1991.

[9] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning Internal Representations by Error Propagation. Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1: Foundations, MIT Press, Cambridge, MA, 1986.

[10] D.W. Patterson, Artificial Neural Networks - Theory and Applications. Prentice Hall, 1996.

[11] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Fiannery, Numerical Recipes. Cambridge Univ. Press, 1992.

[12] A.S. Glassner, "Space Subdivision for Fast Ray-Tracing," IEEE Computer Graphics and Applications, Vol. 4, No. 10, pp. 15-22, 1984.

[13] N. Stolte, A. Kaufman, "Parallel Spatial Enumeration of Implicit Surfaces Using Interval Arithmetic for Octree Generation and its Direct Visualisation," IS98-3^{rd} International Workshop on Implicit Surfaces, pp. 81-87, 1998.

[14] A. Sherstyuk, "Fast Ray Tracing of Implicit Surfaces," IS98-Third International Workshop on Implicit Surfaces, 1998.

[15] J.M. Snyder, "Interval Analysis for Computer Graphics," SIGGRAPH 92 Proceedings, pp. 121-130, 1992.

[16] D. Kalra, A. Barr, "Guaranteed Ray Intersections with Implicit Surfaces," SIGGRAPH 89 Proceedings, pp. 297-306, 1989.

[17] A.R. Conn, N.I.M. Gould, P.L. Toint, Trust-region methods. MPS-SIAM Series on Optimization, 2000.

[18] G. Zhou, J. Si, "Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency," IEEE Transactions on Neural Networks, 9(3), pp. 448-453, 1998.

[19] G. Zhou, J. Si, "Subset based training and pruning of sigmoid neural networks," Neural Networks, 12(1), pp. 79-89, 1999.

[20] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, T.R. Evans, "Reconstruction and Representation of 3D Objects with Radial Basis Functions," SIGGRAPH 2001 Proceedings, pp. 67-76, 2001.

[21] A.P. Witkin, P.S. Heckbert, "Using Particles to Sample and Control Implicit Surfaces," SIGGRAPH 94 Proceedings, pp. 269-277, 1994.