Université d'Ottawa • University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Ana-Maria CRETU

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. A. Sc. (Electrical Engineering)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Neural Network Modeling of 3D Objects for Virtualized Reality Applications

E. Petriu

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

R. Goubran

N. Georganas

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Neural Network Modeling of 3D Objects for Virtualized Reality Applications

by

Ana-Maria Cretu

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

in partial fulfillment of the requirements for the degree of

Master in Applied Science

Ottawa-Carleton Institute of Electrical and Computer Engineering

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

December 2003

Your file Votre référence
ISBN: 0-612-90052-5
Our file Notre référence
ISBN: 0-612-90052-5

I hereby declare that I am the sole author of this thesis.

I authorize The University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Ana-Maria Cretu

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Ana-Maria Cretu

To my dear mother and father
To Pierre

Contents

List of Figures

List of Tables

Abbreviations

ART	Adaptive Resonance Theory
BSP	Binary-Space Partitioning Tree
IFOSART	Incremental Fully Self-Organized Simplified ART
CAD	Computed Aided Design
CSG	Constructive Solid Geometry
CT	Computed Tomography
MLFF	Multilayer Feedforward (Neural Network)
NURBS	Non-Uniform Rational B-Splines
SOM	Self-organizing map
TRPY	Translation-Rotation-Pitch-Yaw (Homogeneous Matrix)

Abstract

This thesis presents a critical comparison between three neural architectures for 3D object representation in terms of purpose, computational cost, complexity, conformance and convenience, ease of manipulation and potential uses in the context of virtualized reality. Starting from a pointcloud that embeds the shape of the object to be modeled, a volumetric representation is obtained using a multilayered feedforward neural network or a surface representation using either the self-organizing map or the neural gas network. The representation provided by the neural networks is simple, compact and accurate. The models can be easily transformed in size, position (affine transformations) and shape (deformation). Some potential uses of the presented architectures in the context of virtualized reality are for the modeling of set operations and object morphing, for the detection of objects collision and for object recognition, object motion estimation and segmentation.

Acknowledgements

I would like to express my great appreciation to my thesis supervisor Dr. Emil Petriu, for being my teacher, my advisor, my mentor and my friend. His vision, experience and guidance helped me find, focus and learn a lot on an exciting research topic. My appreciation goes as well to my co-supervisor Dr. Gilles G. Patry. I would also like to thank to Dr. Nicolas D. Georganas and Dr. Rafik Goubran for their valuable comments regarding the thesis work.

I am truly grateful to my beloved parents for the myriad of ways in which, throughout my life, they have actively supported me in making my dreams and aspirations come true and for their constant, unconditional support and encouragement from the very beginning of my studies. Despite of living miles away, you are constantly in my thoughts and your love and support has been a major stabilizing force over these past two years. Your unquestioning faith in me and my abilities has helped to make all this possible and for that, and everything else, I thank you.

Out of all my heart, I owe my loving thanks to the dearest man in my life that urged me on by way of his untiring support and seemingly unlimited belief in me. Thank you Pierre for your unfailing patience and help in making this thesis come to an end. You consistently transformed the thesis "blues" into the confidence and motivation I needed to complete this work. Your unlimited love and understanding have been my greatest source of strength for everyday life and work and have helped me overcome all hardships throughout this year.

Special thanks to my dear friend Dorina, for the warm-hearted encouragements and valuable advises. All my gratitude to Corina and Voicu for the love, help and support in the beginning of this new life.

Finally I would like to thank to Isabela and Codrin for their kind support and all my friends in the SMR and Discover labs.

Chapter 1

Introduction

1.1. The Context of Virtualized Reality

A *virtualized reality environment* differs from a virtual reality environment in the manner the virtual world models are constructed. While virtual reality is typically constructed using simplistic CAD models and lacks fine details, virtualized reality starts from the real world scene and virtualizes it [1,2]. In other words, a virtualized environment is a conformal representation of the mirrored real world based on information about the real world objects and phenomena as captured by a variety of sensors [3] that preserves the visible detail of the described real-world scene [2]. The virtual navigation of such an environment requires a complete 3D model of the scene [4]. One of the most challenging steps in the process of building a virtualized reality environment is that of modeling the real objects for incorporation into the virtualized world.

Depending on the complexity of the environment and on the application, achieving realism and geometrical correctness for the modeled objects can become a very difficult task. There are several aspects that have to be considered when building an object model, including the methods to define and acquire or create the data that describes the object, the purpose of the model, its computational cost, the complexity of the resulting model, its conformance and convenience, and the ease of model manipulation. Often, information from various types of sensors has to be integrated and registered. The resulting object models are discrete and often too large for real-time interaction. The effectiveness of these models depends strongly on the accuracy of data and on the number of samples they use, and their quality and degree of the approximation can be only determined by validation against experimental results [3].

Much research effort has been made in the area of 3D object modeling over the last decades. The most common approaches are surface modeling, solid modeling and procedural modeling. Recent years have witnessed a new direction of research in object modeling field: neural networks. The strength of neural networks stems from their intrinsic non-linearity, their

computational simplicity and the ability to learn, therefore to generalize. A neural network has the capability to learn high-dimensional, redundant mappings from a limited set of measured data without a prior mathematical model. Its power comes from the collective behavior of the simple neurons. The computational load is evenly distributed across many simple processing elements, thus ensuring a high degree of fault tolerance. Neural networks have the built-in capability to adapt their weights to changes in the environment and are able to provide instantaneously an estimation of the output values for input values that were not part of the initial training set, due to their continuous memory. Although they may take a relatively long time to learn, the recall phase, which is what actually counts in virtualized reality environment applications is done in much less time [3].

Used either as a standalone tool or as complement to classical modeling approaches, neural networks have been applied in several areas of computer graphics for data fusion [5, 6] and improvement of measured data using *a priori* knowledge [7, 8], projection, visualization and exploration of high-dimensional data [9, 10, 11], reconstruction of surfaces and shapes from scattered data, range data and shape from shading [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26], object representation and modeling [27, 28, 29, 30, 31, 32, 33, 34, 35, 36], modeling of isosurfaces [37] segmentation [30, 38], shape matching [39] and volume matching [40], as well as object classification [15, 16, 41] and object motion [15,16, 42, 43, 44].

1.2. Analyzed Structure

The goal of this thesis is to evaluate the representational power of neural networks in the context of 3D object modeling. The emphasis is on particular architectures, their characteristics and potential and not on the real-time behaviour in this stage of the work.

The architectures used in this thesis consist of two cascaded modules: a transformation module and a representation module, as depicted in Fig. 1.1. The representation module is either a multilayer feedforward neural network or one of two self-organizing architectures: self-organizing map and neural gas network. In the first case, starting from a pointcloud that embeds the shape of the object to be modeled, a volumetric representation is obtained using the multilayer feedforward network. The surface of an object is described by a set of zeros in the output response of the network, while the interior or exterior of an object are mapped to a negative or positive value, respectively. The representation is simple, compact, and accurate (as long as the user is willing to pay the computational time for the training phase). It can offer theoretically infinite resolution and saves storage space. The representation is continuous and permits an extensive study of not only the modeled object, but of the entire object space. Thus it inherits the

advantages of the polygonal and surface models, without inheriting their drawbacks. It can be used to perform simple operations such as object morphing, set operations and object collision detection.

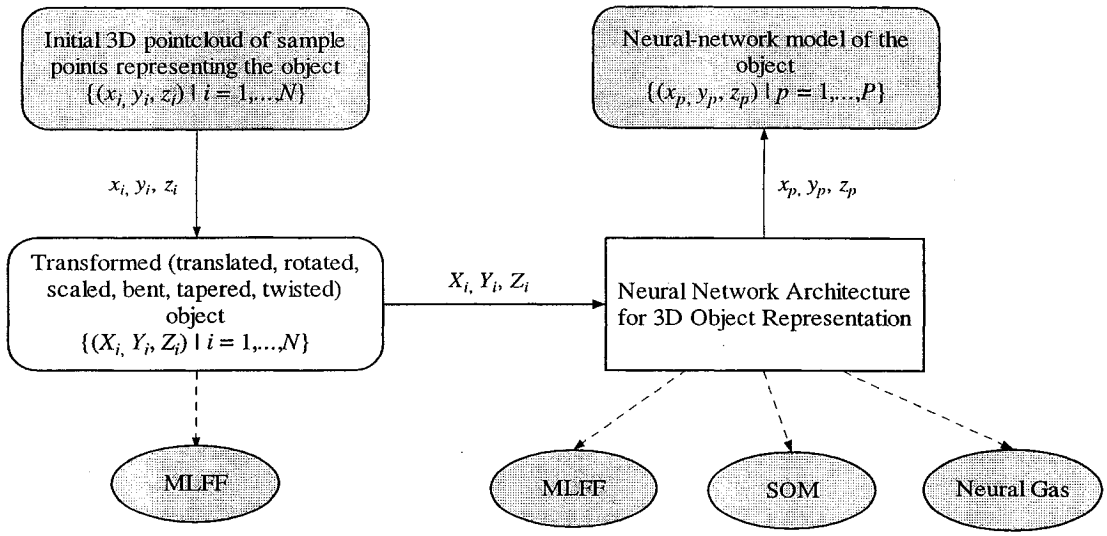


Fig. 1.1. Modeling framework

On the other hand, the representation offered by the two self-organized architectures has the characteristics of a surface model. The models are obtained faster and can achieve higher accuracy than the multilayer feedforward counterpart. Both can be used for object segmentation in complex scenes.

The representation module can be cascaded with a transformation module, whose role is to ensure an easy manipulation of the represented objects in size, position (it models affine transformations) and shape (it also models deformations). Moreover, the module can be used to learn transformation/deformation information, when it is provided with a reference object and a transformed/deformed object. This makes it useful in motion and alignment estimation. When cascaded with the transformation module, the multilayer feedforward architecture can be used for object recognition. In all of the three cases, the representation module cascaded with the transformation module can be used for motion estimation.

1.3. Contributions

The contributions of the thesis can be summarized as follows:

a.) Implementation, analysis and comparison of three neural architectures for object representation in terms of modeling power, computational cost, complexity, conformance, convenience and ease of manipulation.

b.) Study of applications of the neural architectures in the virtualized reality framework.

Moreover, the thesis pursues the development by:

c.) An extension to the 3D case of a multilayer feedforward neural network architecture for volumetric description of 3D objects proposed by Kumazawa [32, 33].

d.) A neural architecture to generate and recuperate parameters of deformed 3D objects represented by neural networks.

1.4. Thesis Organization

The second chapter provides an overview of 3D object modeling approaches. It begins by briefly describing the issues in 3D object modeling and their impact on the quality of the resulting models. Then it looks in depth on classical 3D object modeling techniques and what they involve. It discusses advantages and disadvantages and gives possible applications for each described method. The presentation is not meant to be exhaustive but rather aims at providing a background for the study of neural networks in the context of 3D object modeling. The chapter ends up by discussing several neural architectures and examining their use in object modeling.

The third chapter discusses implementation issues of a multilayer feedforward architecture for 3D object representation. It starts by giving a quick overview on neural networks, their learning ability and the critical factors for a successful learning procedure. It then describes the proposed architecture, the data sets used for training, the training algorithm and the testing strategies. Finally it examines possible applications of the proposed architecture in the virtualized reality framework.

The fourth chapter describes two self-organized architectures for 3D object representation: the self-organizing map and the neural gas network. It starts by providing a brief introduction to the main ideas behind self-organization and links them with vector quantization methods. It then takes a look both to the self-organizing map and neural gas network, taking into consideration aspects such as: purpose, topology, training algorithms, data sets and testing procedures. It finally shows some possible applications of the described models.

The purpose of the fifth chapter is to present the experimental results for the three neural network architectures presented in the previous two chapters. It discusses implementation issues, parameters adjustments and the modeling results in terms of purpose, computational cost, complexity, conformance and convenience and ease of manipulation. It shows how the models can be used in the virtualized reality context. The chapter ends up with a critical comparison between the three neural architectures. The last chapter derives the conclusions of the work and shows possible future developments of the examined topic.

Chapter 2

3D Object Modeling - Issues and Techniques

This chapter provides an overview of 3D object modeling approaches. It begins by briefly describing the issues in 3D object modeling and their impact on the quality of the resulting model. Then it looks in depth on classical 3D object modeling techniques and what they involve. It discusses advantages and disadvantages and gives possible applications for each described method. The presentation is not meant to be exhaustive but rather aims at providing a background for the study of neural networks in the context of 3D object modeling. The chapter ends up by discussing several neural architectures and examining their use in object modeling.

2.1. Issues in Object Modeling

The closest definition of "modeling" to the context of 3D graphics and therefore to the purpose of this thesis can be expressed as: the process of giving a "schematic description" of the objects that compose a scene, "that accounts for their known or inferred properties and may be used for further study of their characteristics"[45]. This is however, not an easy task. The structure inherent in 3D object scenes is difficult for people to understand, and is also difficult for user interfaces to display and manipulate. The big challenge in 3D graphics is "to make simple modeling easy and complex modeling accessible" [46].

There are several aspects that have to be considered when building an object model, that contribute all together to the quality of the final result. They include the methods to acquire or create the data that describes the object, the purpose of the model, the implementation complexity, its computational cost, its conformance and convenience, and the ease of model manipulation.

2.1.1. Methods to Define an Object

There are a numerous ways in which objects can be defined: directly giving the geometric description (e.g. the equation that describes the object), writing programs to create data or using modeling programs, using reconstruction from photographs or using real data in form of clouds of points (a number of points with only their respective (x, y, z) coordinate triplets) obtained from range scanners, optical fringe patterns, mechanical coordinate measuring devices or Computed Tomography (CT) scans. In short, any function or procedure that provides 3D surface data can provide a basis for modeling. However, the way of processing the data might differ for different sources, and thus needs to be taken into account at modeling time.

2.1.2. Purpose of the Object Model

There are two main reasons for building 3D objects: "image making" and simulation [47]. While "image making" requires that the model looks good (is realistic, sufficiently complex and conforms to the geometry of the model), simulation requires accurate models. While the first one is described as interactive and approximate, the other one is generally slow but precise. The future use of an object model is thus an important consideration when choosing a certain modeling approach.

2.1.3. Computational Cost

Another important aspect to be taken into account is, knowing the needs of an application, what price is the user willing to pay in terms of computational costs. The computational cost of a model may be gauged in terms of computer storage space, object construction time, display time, or in application use [47]. The storage space of a model refers to the space required to store the primitives that represent it. Judging the model in terms of object construction time means evaluating the model for the quantity (how much time does it need to be processed) and quality of the source data (how much time is needed to remove errors). The display time refers to the cost of object display, including time and image quality issues. Often, a real-time behavior of an object model is required.

2.1.4. Complexity

Another question to be answered when evaluating an object model's quality is how complex the model is. The complexity of a model may be judged as a trade-off between the number of primitives and the inherent complexity of the individual primitives. There are objects

that are composed of many simple forms or objects that are inherently complex. For example, man-made objects and mechanical parts can be seen as a set of simple forms, while biological and other natural structures are inherently complex [47].

2.1.5. Conformance and Convenience

Other qualities that the users look for in a model are its conformance and convenience. Conformance measures the degree to which the model actually represents the desired objects and provides a realistic, geometrically correct representation for the modeled object. Usually, the quality and the degree of the approximation of the model can be determined only by validation against experimental results. The convenience refers to the ease of performing extensive parametric studies on the described model, in which independent model parameters can be modified, in order to gain global knowledge on the properties of a certain modeled object [3].

2.1.6. Ease of Model Manipulation

Related to convenience, the user wants to gain the global knowledge of the objects' properties in an easy manner. An object model must be easily transformed in position, size or shape, and offer facilities for evaluating its geometric properties. The ability to manipulate an object depends greatly upon its representations. The most common representations of 3D objects are surface models, solid models and procedural models. The main ideas of the three categories, together with the advantages, disadvantages and most common uses will be further presented.

2.2. Surface Models

Surface modeling techniques represent surfaces explicitly. Points in 3D space are then characterized as either on or off a given surface. Though surfaces can be joined to create the illusion of a solid object, the solidity is not inherent in the nature of the surfaces' descriptions. The most common surface representations are polygonal models and functional surfaces. Functional surfaces can either be explicit (parametric curves and surfaces) or implicit functions.

2.2.1. Polygonal Techniques

Polygonal models are one of the most commonly encountered representations in computer graphics. The models are defined as sets of connected polygons, called *meshes*. The polygons are sized, shaped, and positioned so that they completely tile the required surface at some resolution. Each polygon consists of some connected vertex, edge, and face structure.

An important thing to consider when modeling with polygons is the integrity of the polygons. Nonplanar polygons create holes in the objects, because the plane is twisted and doesn't render. The solution is to split the nonplanar polygon into three-sided polygons (these cannot become nonplanar) or use subdivision technology to smooth the mesh [49]. Subdivision techniques add polygons to a model, in an effort to smooth rough edges.

Advantages

Polygon models are relatively simple to define, manipulate, and display and are the commonest model processed by workstation hardware and commercial graphics software. They are flexible and allow varied mesh density according to the needs of a certain application. They are usually used to provide a low-resolution model, which can then be transformed into a high-resolution using subdivision.

Disadvantages

The number of polygons needed to accurately define a complex object may be large. This has implications for memory usage and rendering times [50]. Also, smooth curves cannot be easily created with polygons. To create the illusion of smoothness, a large number of polygons needs to be used and numerous points edited [49].

Applications

The ability to work with low-resolution models, create varied density and apply minute details makes polygons the most appropriate technique for character and creature modeling. They are also suited for natural organics and linear models (e.g. furniture, computer equipment, building, city streets) because they do not require precision curves [49].

2.2.2. Parametric Curves and Curved Surfaces

Since polygons are good mainly at representing non-curved surfaces, considerable effort has been made to determine mathematical formulations for curves and curved surfaces. A *curve* can be defined as "the locus of a point moving with one degree of freedom or the locus of a one-parameter family" [52]. A polynomial parametric curve of degree n is of the form:

$$p(u) = \sum_{k=0}^{n} u^k c_k \qquad (2.1)$$

where each c_k has independent x, y, z components and u is the curve parameter. For $0 \leq u \leq 1$ the above equation defines a curve segment. If a high degree polynomial is chosen, there will be many parameters that can set the desired shape of the curve, but the evaluation of points on the

curve will be costly. In addition, as the degree of a polynomial becomes higher, there is more danger that the curve will become rougher [53] and will need more computation time. A too low degree, on the other hand, will give too little flexibility in controlling the shape of the curve. Although they are still approximations of the real curve, parametric curves are more accurate and easier to manipulate than the polygonal models.

The curved surface is the natural extension of a curve. A *surface* is the locus of a point moving with two degrees of freedom and its polynomial parametric representation is:

$$p(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} u^i v^j c_{ij} \tag{2.2}$$

A *surface patch* is defined for $u, v \in [0, 1]$. Patches are joined along their boundary edges into more complex surfaces. The shape of a patch is derived from control points or tangent vectors. The parametric form of curves and surfaces is the most flexible and robust for computer graphics [53].

There are many formulations of curves and curved surfaces, including cubic Hermite, Bezier, rational Bezier, splines, B-splines, NURBS, beta-splines and thin-plate splines, just to mention a few. Some of them will be further presented.

2.2.2.1. Bezier Curves and Patches

Together with cubic Hermite curves and patches, Bezier curves and patches were among the earliest forms investigated and used in geometric modeling. Their relative simplicity and natural versatility make them good starting points for studying curves and surfaces [52]. The *Bezier curve* for $n+1$ control points can be described recursively with:

$$p_i^k(u) = (1-u)p_i^{k-1}(u) + up_{i+1}^{k-1}(u), \quad \begin{cases} k = 1..n \\ i = 0,...,.n-k \end{cases} \tag{2.3}$$

where p_i^k are the intermediate control points obtained after the linear interpolation has been applied k times and p_i^0 are the initial control points. A Bezier curve can also be described with an algebraic formula, called the *Bernstein form*:

$$p(u) = \sum_{i=0}^{n} B_i^n(u)p_i, \quad B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}, B_i^n(u) \in [0,1], \quad when\, u \in [0,1], \quad \sum_{i=o}^{n} B_i^n(u) = 1 \tag{2.4}$$

Advantages

The entire Bezier curve will be located in the convex hull of the control points. This is a useful property when computing a bounding area or volume for the curve. The Bezier control points give a good indication of the shape of the corresponding curve [54], which can be adjusted, just by moving these points. Moreover, instead of computing points on a Bezier curves, and then rotating the curve, the control points can be rotated first, and then the points on the curve can be computed. This is much faster than doing the opposite. Bezier curves are the easiest of the curve forms to subdivide [52].

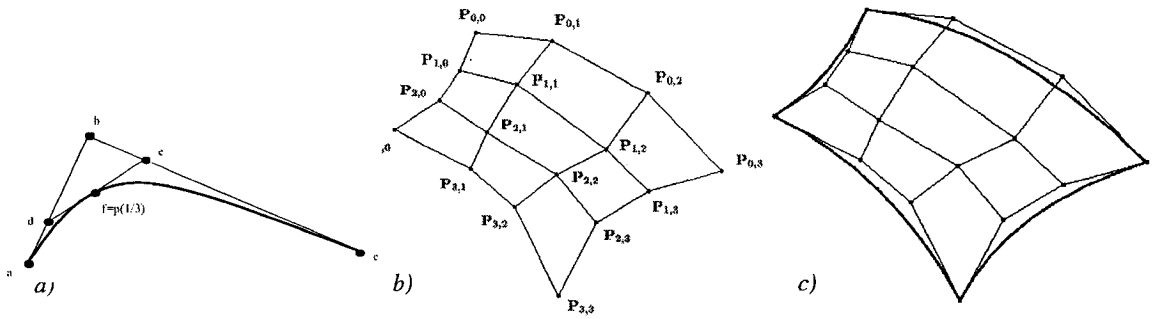


Figure 2. 1. (a) Bezier Curve. (b) Bezier control points and (c) the corresponding patch

Disadvantages

More control over the curve means increasing the degree of the polynomial and making the evaluation more expensive [54]. Moreover, Bezier points are global in nature. Each point has an effect on the entire parameter interval, such that changes in it induce a change in the entire curve [53, 52]. Bezier curves do not have many degrees of freedom and also, not every curve can be described by a Bezier curve [55]. Last but not least, a Bezier curve is not invariant under perspective transformation. Another alternative to solve some of the above-mentioned problems is to use a *rational Bezier curve.*

$$p(u) = \frac{\sum_{i=0}^{n} w_i B_i^n(u) p_i}{\sum_{i=0}^{n} w_i B_i^n(u)} \tag{2.5}$$

For a rational Bezier curve, the same set of control points can produce many different shapes, interpolating the same endpoints and with the same end tangents. Changing one or more of the weights changes the shape of the curve. This means that a rational cubic Bezier curve has four more degrees of freedom than the non-rational cubic counterpart [52]. The rational Bezier curve is invariant under a perspective transformation. If its control points are subjected to this transformation, then all other points are produced as usual from the transformed control points. The resulting curve is the accurate perspective transformation of the original. Moreover, using the

rational Bezier formulation we can accurately represent conic curves, which the non-rational counterpart cannot do. The main justification for using rational curves in practice is their ability to represent conics and circles exactly. Their use is mainly limited for that purpose.

The Bezier curve can be extended to a patch by introducing one more parameter in eq. (2.3). The surface representations exhibit advantages and disadvantages similar to those of the corresponding curves.

2.2.2.2. Cubic Hermite Curves and Patches

The cubic Hermite curve is defined by starting and ending points (p), and starting and ending tangents (m) [51], as in Fig. 2.2.

The cardinal form of a Hermite curve [56] is:

$$p(u) = p_0 H_0^3(u) + m_0 H_1^3(u) + m_1 H_2^3(u) + p_1 H_3^3(u) \tag{2.6}$$

where the cubic Hermite polynomials are given by:

$$H_o^3(u) = B_0^3(u) + B_1^3(u),\ H_1^3(u) = 1/3\, B_1^3(u),\ H_2^3(u) = -1/3\, B_2^3(u),\ H_3^3(u) = B_2^3(u) + B_3^3(u) \tag{2.7}$$

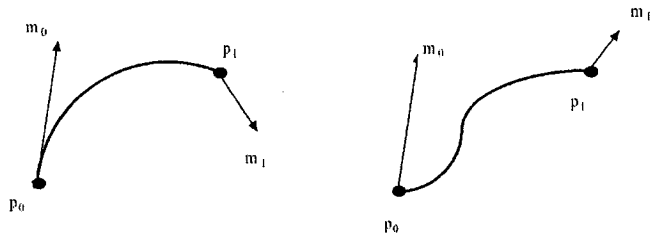


Figure 2. 2. (a) Hermite curves (redrawn from [51])

and B are defined as in eq. (2.4).

The curve interpolates all its control points and is easy to subdivide [52]. It is invariant under rotation, scaling and translation, but is not invariant under perspective projection [48]. When interpolating between more than two points using Hermite curves, a mean to control the shared tangents is needed. *Kochanek-Bartels curves* [57] give the user three intuitive parameters to control shared tangents – bias (influences the direction of the tangent), tension (modifies the length of the tangent vector) and continuity (controls behaviour at the joints).

Like the Bezier curve, a Hermite curve can be extended to a Hermite patch by adding one more parameter in eq. (2.6).

2.2.2.3. Splines

The other major category of curves and surfaces used in computer graphics are splines. Though many different formulations of spline exist (e.g. natural B-splines, beta-splines [48, 52], thin-plate splines [58, 59]), the most commonly used are B-splines and NURBS. A *B-spline curve* is a generalization of a Bezier curve. Given $n + 1$ control points $c_0, c_1, ..., c_n$ and a knot vector $U = \{ u_0, u_1, ..., u_m \}$, the B-spline curve of degree p defined by these control points and the knot vector U [52] is:

$$p(u) = \sum_{i=0}^{n} N_{i,p}(u) c_i N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases}, \quad N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.8}$$

where $N_{i,p}(u)$ are called B-spline basis functions of degree p.

Advantages

B-spline curves satisfy all the important properties that Bezier curves have. However, they provide more control flexibility than Bezier curves can do. They are defined locally and changes in one of the points of the associated control polygon affect the curve locally. It is also possible to add new control points without increasing the polynomial degree, and even to change the order of continuity between neighbouring spline segments by choosing the control points appropriately [54]. A valuable property is that they are invariant under affine transformations [52]. Models generated with splines theoretically offer infinite resolution [49].

Disadvantages

B-spline curves require more information (the degree of the curve and a knot vector) and a more complex theory than Bezier curves. From computer graphics perspective, B-splines cannot be used to model details. They cannot accurately represent small bumps, wrinkles, detail branching or complex objects with many protrusions [49]. Moreover, they cannot represent conic sections (circles, ellipses, spheres, cylinders).

Applications

While not suitable for modeling details and protrusions, splines are to be used when exact curves are needed to describe the physical surface of an object or for prototyping low-detail models.

A *NURBS curve* of degree m is given by:

$$p(u) = \frac{\sum_{i=0}^{n} w_i N_i^m(u) p_i}{\sum_{i=0}^{n} w_i N_i^m(u)}, \quad N_i^m(u) = \frac{u - u_i}{u_{i+m} - u_i} N_i^{m-1}(u) + \frac{u_{i+m+1} - u}{u_{i+m+1} - u_{i+1}} N_{i+1}^{m-1}(u), \; m \ge 1, i = 0..n \tag{2.9}$$

where

$$N_i^0(u) = \begin{cases} 1 & for\ u_i \le u \le u_{i+1} \\ 0 & otherwise \end{cases} \tag{2.10}$$

is the normalized B-Spline basis function defined recursively, p_i are control points for $i = 0..n$, w_i are weights and $U = \{ u_0, u_1, ..., u_m \}$ is the knot vector. The NURBS curve can also be extended to a NURBS surface.

Advantages

The NURBS curves and surfaces can be easily editable by adjusting the knots. The resulting models are efficient in terms of memory use. Very little memory is needed to store quite complex curve structures [50]. NURBS provide theoretically infinite resolution. They allow addition of points to the curve without altering its shape. Unlike parametric curves and natural B-splines, they are able to match conic sections with a minimum of values. The perspective division embedded in the construction of NURBS curves ensures that they are handled correctly in perspective views [53].

Disadvantages

Like B-splines, NURBS are not good for modelling details, holes and branches. A great number of steps is required between constructing a curve network and actually getting the visual feedback of a renderable object. It is difficult to see which control point will deform which curve while adjusting control points [49].

Applications

NURBS are well suited to exact curves modelling and as a starting point for polygonal modeling.

2.2.3. Implicit Surfaces

Instead of using parameters such as (u, v) to explicitly describe a point on the surface, an implicit function of the form:

$$f(x, y, z) = f(p) = 0 \tag{2.11}$$

can be used. In other words, a point p is on the implicit surface if the result is 0 when the coordinates of the point are inserted into the implicit function f.

Advantages

A main advantage of implicit representation over its parametric counterpart is that they comprise more information about their interior and exterior. Implicit surfaces are inherently

manifold. They can be collided and deformed [61]. Moreover for the same polynomial of degree n, implicit algebraic surfaces have more degrees of freedom, compared with rational parametric surface of the same degree. Hence, implicit algebraic surfaces are more flexible to approximate a complicated structure with fewer number or pieces or to achieve a higher order of smoothness [62].

Disadvantages

Despite their many advantages, implicit surfaces are difficult to render efficiently. Today's real time graphics systems are heavily optimized for rendering triangles, therefore an implicit surface should be converted to a mesh of triangles before being rendered [63]. The representation being multivalued may cause the real zero contour surface to have multiple sheet, self-interactions and several other undesirable singularities [61]. Their use is limited by the difficulty in identifying specific locations on a surface.

2.2.3.1. Quadrics and Superquadrics

Quadrics and superquadrics are the most important case of implicit functions. The class includes ellipsoids, parabaloids, and hyberboloids [64]. A quadric is described by the equation:

$$q(x, y, z) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + a_{33}z^2 + 2a_{23}yz + 2a_{13}xz + b_1x + b_2y + b_3z + c = 0 \qquad (2.12)$$

Quadrics are a convenient representation for spheres, ellipsoids and cylinders. They are used in molecular modeling and solid modeling systems [48].

The *superquadrics* are a mathematical generalization of quadrics, which include an interesting class of shapes within a single framework: spheres, ellipsoids, and objects that arbitrarily closely look like prisms, cylinders, stars or toruses (Fig. 2.3). A superquadric is described by the following implicit function:

$$\left(\left(\frac{x}{a_1}\right)^{2/\varepsilon_2} + \left(\frac{y}{a_2}\right)^{2/\varepsilon_2}\right)^{\varepsilon_2/\varepsilon_1} + \left(\frac{z}{a_3}\right)^{2/\varepsilon_1} - 1 = 0 \qquad (2.13)$$

where a_1, a_2, a_3 define the superquadric size in x, y, z coordinates and ε_1 and ε_2 are the squareness parameters in latitude and longitude plane, respectively.

Advantages

The shape of the superquadrics is controlled by simple parameters (ε_1 and ε_2). Their modeling capabilities can be further enhanced by deforming them in different ways, by tapering, bending or twisting (presented in 2.3.4) [65, 66].

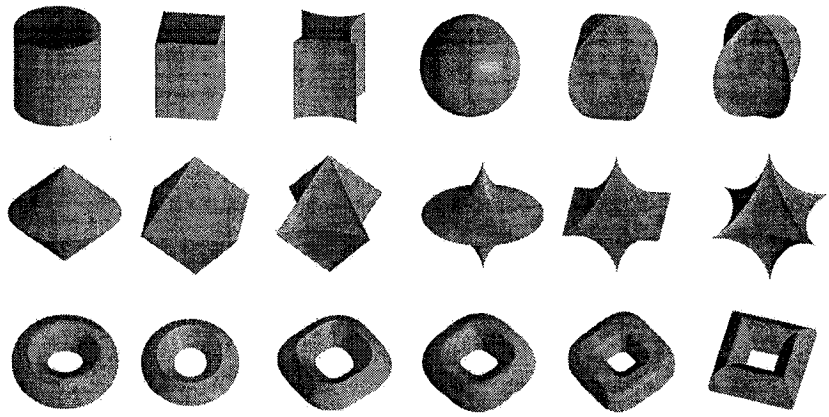


Figure 2. 3. Superquadrics – Superellipsoids and supertoruses

Disadvantages

However, superquadrics lack the descriptive ability to effectively capture local surface shape changes on freeform objects [67]. They have problems in modeling linear objects, mechanical organics and models that require close-ups. The representation is nonunique, which might cause problems in object matching with a database of models [66].

Applications

Superquadrics are well suited for modeling complex organics and their capability of being easily deformed makes them a good choice for animation. They have been used to recognize a large class of objects [30, 66] and to build composite parts using unions of several superquadrics.

2.2.3.1. Metaballs

Another case of implicit functions is the metaballs. *Metaballs* consist of a series of components, spherical in shape, each associated with a size and an "attractive" or a "repulsive" strength. The final shape of the metaball is computed based on the position, size, and strength of each component. The center of the metaball has the maximum density value, and the density decreases outwards. Metaballs are spheres that blend together to form a single object [49].

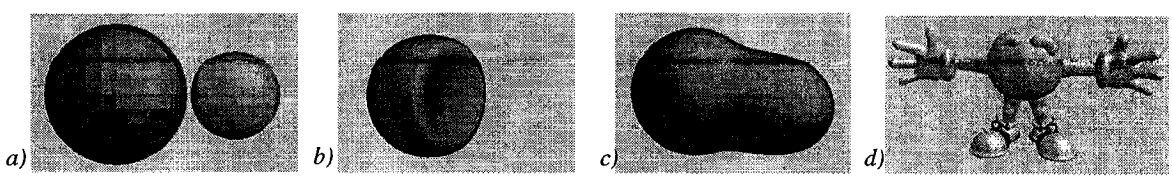


Figure 2. 4. (a) Metaballs. (b,c) Negative and positive metaball influence.
(d) Metaball model (reproduced from [49])

Advantages

Metaballs editing is easy because of the ability to modify the spherical shape of the metaball (they stretch, shear, bend and twist). They are well suited for modeling complex, highly detailed organics. Metaball modeling is one of the fastest modeling methods available. It is at the same time well suited for animation, in modeling water effects and liquid metals [49].

Disadvantages

The determination of the correct placement of the metaballs is a challenge. Quite often the metaballs need to be separated by a gap to produce the proper effect. Metaballs models can result in disorganized meshes and models tend to look as if they are made of spheres. Moreover, small details are very difficult to obtain with metaballs.

Applications

While well suited for detailed creatures and characters, natural organics and rapid prototyping, metaballs are not to be used for linear models, mechanical organics or models that require close-ups [49].

2.3. Solid Models

Solid modeling describes the volume of space occupied by a solid. It differs from surface modeling in several ways. Solid objects are, unlike surfaces, closed (contain their boundary). While it is possible to decompose a solid object into smaller patches, it is not possible to convert a surface into a "real" 3D object. Moreover, unlike surfaces, a solid object has an inside and an outside.

The solid representation schemes can be categorized into implicit, enumerative, boundary schemes [69] and deformational schemes.

2.3.1. Implicit Schemes

Implicit representations give rules for testing which points belong to an object and which do not. The most important representations comprised in this category are the pure primitive instancing schemes, constructive solid geometry and sweep presentations.

2.3.1.1. Pure Primitive Instancing

In a pure primitive instancing scheme each object family is called a *generic primitive,* and individual objects within a family are called *primitive instances* [68]. A generic primitive is a 3D solid shape specific to the application area [52] and represented by fixed-length tuple of values [68], such as ('PARALLELEPIPED', *l, w*), where 'PARALLELEPIPED' represents the

name of the family and l is the length of the sides and w the width. The primitive instance is a linear transformation of an existing generic primitive. The instances are not usually combined to form more complex shapes, though no theoretical barriers prevent it.

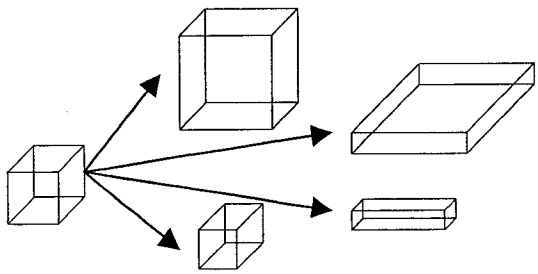


Figure 2. 5. Instances of the unit cube (redrawn from [52])

Advantages

Pure primitive instancing schemes are easy to validate, unambiguous, unique, concise, and easy to use.

Disadvantages

One disadvantage is the lack of means for combining instances to create new and more complex objects. Furthermore, it is difficult to write algorithms for computing properties of represented solids. A considerable amount of family-specific knowledge must be built into the algorithms, and therefore each object family must be treated as a special case, allowing no uniform overall treatment [68].

Applications

Often used for relatively complex objects, in manufacturing world, such as gears and bolts, that are tedious to define in terms of boolean operations [48].

2.3.1.2. Constructive Solid Geometry

Constructive Solid Geometry is a set of modeling methods to define complex solid using closed primitives, rigid body motions and set operations [68, 69]. Rigid body motions are described in terms of TRPY convention by:

$$
\begin{aligned}
X &= a_1 x\cos\theta\cos\varphi + a_1 y(\cos\theta\sin\varphi\sin\psi - \sin\theta\cos\psi) + a_1 z(\cos\theta\sin\varphi\cos\psi + \sin\theta\sin\psi) + x_T \\
Y &= a_2 x\sin\theta\cos\varphi + a_2 y(\sin\theta\sin\varphi\sin\psi + \cos\theta\cos\psi) + a_2 z(\sin\theta\sin\varphi\cos\psi - \cos\theta\sin\psi) + y_T \\
Z &= -a_3 x\sin\varphi + a_3 y\cos\varphi\sin\psi + a_3 z\cos\varphi\cos\psi + z_T
\end{aligned}
\qquad (2.14)
$$

where X,Y,Z are the coordinates of the points in the transformed solid, x, y, z the coordinates of points in the original one, θ defines the rotation around z, φ the rotation around y, ψ the rotation

around x, x_T, y_T, z_T describe the translation and a_1, a_2, a_3 define the scaling factor along x, y, z respectively. The equations above permit the determination of the transformed solid coordinates for pure transformations and combinations of transformations as well.

The set operations used are union, difference and intersection. A regularized union, intersection or difference ensures that the new object is a closed one, which is a must in solid modeling.

The CGS representation is an ordered binary tree with nonterminal nodes operators and terminal nodes representing either primitive or transformation leaves which contain the arguments of rigid motions. The operators can either be rigid motions or regularized union, intersection, or difference.

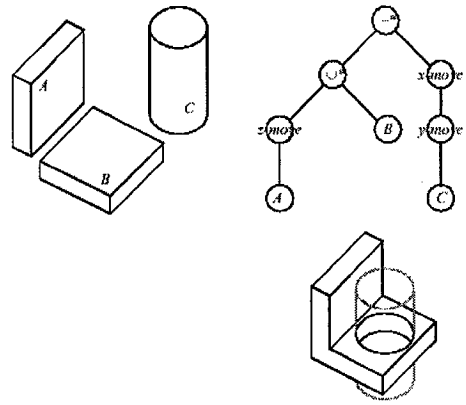


Figure 2. 6. A CSG model and its tree structure [72]

An object therefore exists as a tree structure (Fig. 2.6), which is evaluated during rendering [47].

Advantages

The CSG representation is unambiguous and concise. The set operations are computationally convenient and the rigid motion parameters offer a natural control of the solid's shape, position and orientation. The relatively simple and robust data structures and the elegant algorithms contributed to the popularity of CSG in computer modeling [69].

Disadvantages

The objects that can be modeled with CSG are fairly limited compared to other methods. The lack of explicit representation and parameterization of the interior of the solid object is another drawback, actually associated with all implicit models. Comparison of solids is also problematic, since the CSG representation is non-unique. Moreover, the display is usually expensive.

Applications

CSG is used in those applications where simple geometric objects are desired, or where mathematical accuracy is important.

2.3.1.3. Sweep Presentation

Sweep presentations are constructed by moving a curve, surface, or solid along some path. The locus of points generated by this process defines a new 2D or 3D object. A translational sweep (extrusion) is defined by moving a planar curve or planar shape along a straight line normal to the plane of the curve, the former generating a surface and the latter a solid (Fig.2.7a). A rotational sweep is defined by rotating a planar curve or shape with finite area about an axis (Fig. 2.7b). A general sweep is one whose generating shape follows some arbitrary curved path, and which itself may change size, shape and orientation [52].

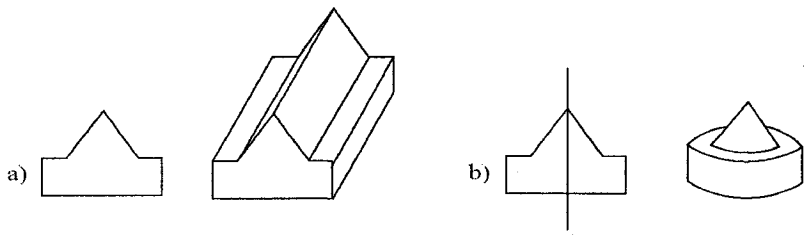


Figure 2. 7. (a) Translational sweep presentation, (b) Rotational sweep presentation

Advantages

Sweep representations are unambiguous representations, simple to understand and use. Only a small data set is needed to specify the shape. They are mathematically concise. Occlusions can be handled by storing a sweep presentation of the free space instead of the solid objects.

Disadvantages

Their use is restricted to objects with rotational or translational symmetry. Moreover, the sweep presentation is not unique. This problem can be solved using *generalized cylinders*, a combination of sweep presentation and contours. This definition ensures that the representation is unique over a class of transformations [68].

Applications

Sweep presentations can accurately represent a large class of engineering and manufacturing objects and processes. They proved to be practical and efficient for modeling constant cross-section mechanical parts and for simulating and analyzing material-removal operations in manufacturing. They can also be used to detect potential interference between parts

and mechanisms [52]. A moving object collides with a fixed object if the swept volume of the moving object intersects the fixed object.

2.3.2. Enumerative Schemes

A more direct way to define which points belong to the solid and which do not is to enumerate the points by an explicit parametric rule.

2.3.2.1. Parametric Solids

The simplest and most direct mathematical approach to model a solid is by using continuous, three-parameter, single-valued functions of the form $x = x(u,v,w)$, $y = y(u,v,w)$, $z = z(u,v,w)$, where $u, v, w \in [0,1]$. These functions define the coordinates of the set of points comprising the interior and exterior of the solid. This representation is called *parametric solid*, or *hyperpatch* [52]. It inherits the advantages and disadvantages of parametric curves and surfaces. It is unambiguous, unique and concise.

2.3.2.2. Space-Partitioning Solids

An object can be decomposed in parts until its parts are describable. The solid will be thus a union of cells into which the object is divided. This process is known as *cell decomposition*. Space-partitioning representation is a special case of cell decomposition, where the shape of the cell is a cube or parallelepiped (*voxel*) and the cells are fixed in a spatial grid. As the size of the cube decreases, this method approaches the representation of a solid body as a set of contiguous points in space [52].

Voxels

The simplest and convenient way to represent a solid is as an ordered list of voxels occupied by the solid, called *spatial array* [68].

Advantages

A spatial array is an easy to validate, unambiguous and unique representation. The complexity of modeled objects is the same, in that the representation uses the same amount of computer memory no matter what physical size it is representing. Set operations are easy to implement and the representation is affine invariant.

Disadvantages

However, the voxel representations can be quite verbose and require large amounts of data storage. The representation is not very accurate and is not concise.

The display is usually extensive.

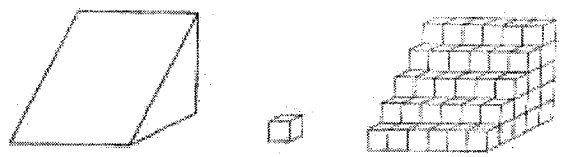


Figure 2. 8. An object and its voxel representation (reproduced from [70])

Applications

Spatial arrays have been successfully used in certain architectural applications where buildings are sufficiently modular or in tomography where irregular biological objects are modeled approximately by polyhedra [68].

Octrees

Octrees are an approach to 3D object representation based on the recursive subdivision of an object array into octants.

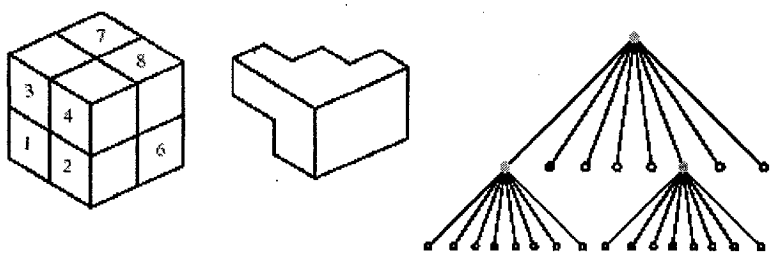


Figure 2. 9. A model and an octree representation (reproduced from [72])

This process is represented by a tree of degree 8, in which the root node represents the entire object with octants labeled. The leaf nodes correspond to those cubes of the array for which no further subdivision is necessary [71]. Leaf nodes are black ('full') or white ('empty'), depending on whether their corresponding cubes are entirely within or outside of the object, respectively. All nonleaf nodes are gray.

Advantages

The octree representation is easy to validate. Set operations are easy to implement due to the simplicity of the tree structure. The complexity of modeled objects is the same, in that the representation uses the same amount of computer memory no matter what physical size it is representing. Because usually the cubes are larger in size than the voxels, the octree is an economical hierarchical representation. Octrees are able to provide a representation for many types of objects and their precision is determined only by the size of the smallest cube [48]. If all

leaves of a particular node are outside the field of view of the viewpoint, that node can be discarded from rendering. This is a fast and effective way of speeding up the rendering engine.

Disadvantages

The octree representation is neither concise, nor affine invariant. The display is expensive and high accuracy implies high storage. Using an octree representation, an object can only be approximated, and not fully represented. This limitation restricts the usability of octrees in the existing graphics systems [48].

Applications

Octrees are excellent at modeling 3D spaces, which contain volumes that are highly connected. They can also be extended to model arbitrarily large and growing surfaces, which do not have a priori bounds [73].

Binary Space-Partitioning Trees

A *binary space-partitioning* tree is a recursive, hierarchical subdivision of the object space. Each subspace is partitioned by hyperplanes that intersects the interior of that subspace and the two resulting subspaces are labeled "in and "out" [48]. These two subspaces can be further partitioned, as depicted in Fig. 2.10.

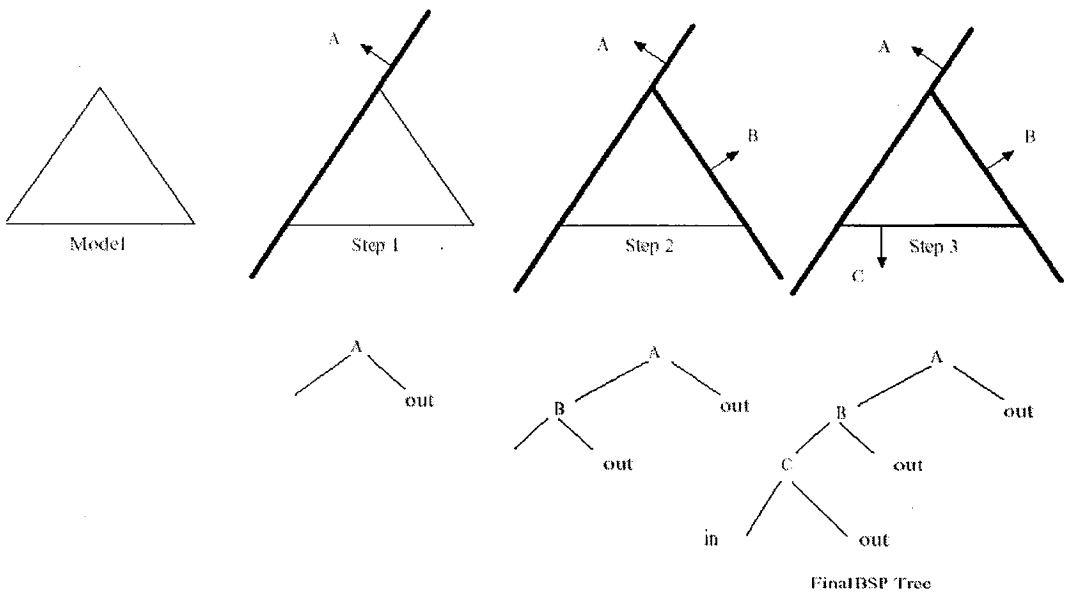


Figure 2. 10. A model and the steps in the BSP tree construction

Advantages

The BSP representation is an easy to validate, unambiguous, simple and elegant representation. It is affine invariant and set operations are easy to be performed. The display is efficient.

Disadvantages

On the other hand, the BSP tree is a non-unique and not concise representation, and its validity is computationally expensive to establish. The representation does not ensure that the result is a bounded solid (e.g. the 3D BSP tree consisting of a single internal node, with "in" and "out" children defines an object that is a half-space bounded by only one plane [48]). In order to add or remove cells from the described solid, a part of BSP tree needs to be reconstructed. This property limits the use of BSP trees to static models.

Applications

BSP trees, as the octrees, are an efficient scheme to be used when a fast rendering is needed. If all leaves of a particular node are outside the field of view of the viewpoint, that node can be discarded from rendering. This is a fast and effective way of speeding up the rendering engine. The tree can also be used for a number of other effects, such as shadows.

2.3.3. Boundary Schemes

Another idea to represent a solid object is to describe the surfaces enclosing that solid. Boundary models are complete representations of a solid as an organized collection of surfaces. The solid is thus a union of faces (surfaces), bounded by edges (curves), which in turn are bounded by vertices (points) [52], as presented in Fig. 2.11.

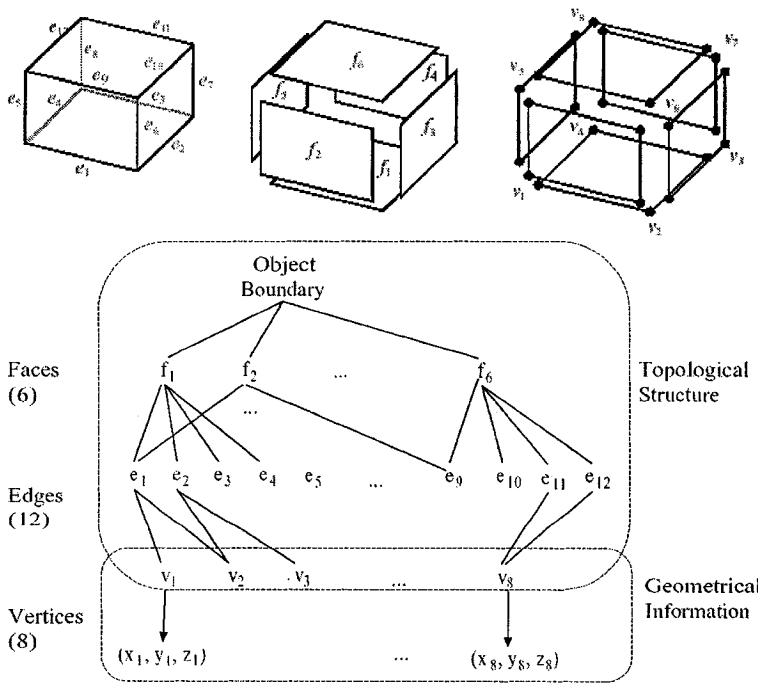


Figure 2. 11. A cube with marked faces, edges and vertices (reproduced from [72]) and its boundary representation

Advantages

The boundary representation is unambiguous. It is as powerful as, or even more powerful than CSG schemes, given that the appropriate primitive surfaces are chosen. The main virtue of boundary representations lies in the ready availability of representations for faces, edges, and the relations between them. Moreover, boundary representation can be much smaller than most 3D cellular representations of the same object.

Disadvantages

Boundary representations are generally not unique and are not concise (usually an order of magnitude longer than corresponding CSG representations). They are not trivial to construct and maintain. There are two types of conditions for validity that a boundary scheme has to satisfy: topological (realizable and orientable 2-manifolds) and geometrical. Geometrical conditions are usually difficult and computationally expensive to check.

Applications

The data on faces, edges and relations between them are important for generating line drawings and graphic displays and for supporting graphic interaction [68]. Thus it is not surprising that boundary representations have been used extensively in computer graphics.

2.3.4. Deformational Schemes

Deformations can be seen as an extension of affine transformations and set operations for solid object modeling. There are two categories of deformations. A *locally specified deformation* modifies only a set of points corresponding to a sub-region of the object surface (the tangent space of the solid is modified [65]). A *globally specified deformation*, in contrast, affects an object as a whole, by explicitly modifying the global coordinates of all solid's points in space [65]. Such transformations include global tapering, global twisting and global bending. Only the latter will be further detailed. Deformations can also be combined in the form of hierarchical structures. Thus they increase the range and complexity of solids that can be modeled.

Global Linear Tapering

The *tapering* deformation can be thought of as a differential scaling, which only changes the length of two components (say y and z), without changing the length of the third (x). Thus, the global tapering along x is given by:

$$\begin{aligned} X &= x \\ Y &= f_y(x)y \\ Z &= f_z(x)z \end{aligned} \tag{2.15}$$

where X, Y, Z are the coordinates of the points in the deformed solid, x, y, z the coordinates of points in the original solid and $f_y(x)$ and $f_z(x)$ are the tapering functions along y and z axes. The tapering functions are piecewise linear functions which decrease as x increases [65]. Tapering along y and z are obtained in the same manner.

Figure 2. 12. (a) Tapering, (b) Twisting, (c) Bending (reproduced from [74])

Global Axial Twisting

A *twist* can be approximated as a differential rotation. Only two components are rotated as a function of height, while leaving the third one unaltered. The global twist around x is produced by the following set of equations:

$$\begin{aligned} X &= x \\ \theta &= f(x) \\ Y &= y\cos\theta - z\sin\theta \\ Z &= y\sin\theta + z\cos\theta \end{aligned} \tag{2.16}$$

where X, Y, Z, x, y, z are defined as before, and the twist proceeds along x at a rate of the derivative of f, $f'(x)$ radians per unit length [65]. Twisting along y and z are obtained in the same manner.

Global Linear Bending

Another deformation consists in simultaneously translating and rotating two components around the third. This deformation is called bending. A bend along x is described by eq. (2.17) [74] where X, Y, Z, x, y, z are defined as before, and k is the bending parameter ($1/k$ gives the curvature of the bending). The bending along y and z are achieved in the same manner as for x.

$$\begin{aligned} \theta &= ky \\ X &= x \\ Y &= -\sin\theta\left(z - \frac{1}{k}\right) \\ Z &= \cos\theta\left(z - \frac{1}{k}\right) + \frac{1}{k} \end{aligned} \tag{2.17}$$

2.4. Procedural Models

Some irregular or complex objects with dynamic topologies, or objects that have no solid surface are difficult to be represented as a set of surface primitives or polyhedral models. Surface and polygonal models usually need explicitly storing of all the requisite data [47]. Moreover, they have difficulty in combining computer graphics with physical laws. Although one can build and animate polygonal models of real-world objects, it is far more difficult to make these graphical objects act as solids and not penetrate one another [53]. Thus, researchers in the domain looked for alternatives. One of the proposed solutions is procedural modeling.

Procedural models are generative processes that describe objects that can interact with external events to modify themselves [48]. Each procedural model of a 3D object is described in terms of components and a procedure (algorithm) that shows how to generate the object and how to control its shape using these components. The necessary data is produced only when requested. The user can control how many primitives are produced and at which point in the generation process. Equally important is that procedural graphics provide an object-oriented approach to building models – an approach that should be of increasing importance in the future [53]. The most common procedural models are fractals, graftals and particle systems.

2.4.1. Fractals and Graftals

The term "fractal" denotes "A geometric pattern that is repeated at ever smaller scales to produce irregular shapes and surfaces that cannot be represented by classical geometry. Fractals are used especially in computer modeling of irregular patterns and structures in nature"[45]. The concept is best illustrated using the 'von Koch snowflake' (Fig. 2.13). The snowflake is built by starting with an equilateral triangle, removing the inner third of each side, building another equilateral triangle at the location where the side was removed, and then repeating the process indefinitely [58].

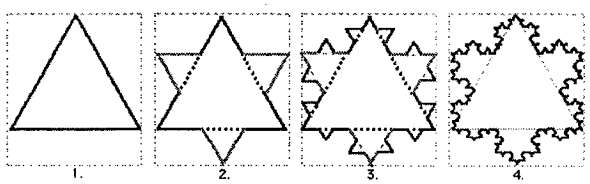


Figure 2. 13. Von Koch Snowflake

Graftals use deterministic processes to model more repetitive patterns [47]. Parallel graph grammar languages (L-grammars), called by Smith [75] 'graftals', were first used to describe the structure of certain plants. The shape of the plant is defined by a string of symbols constructed by

a graftal grammar (Fig.2.14). A graftal grammar consists of an alphabet of symbols, a set of production rules and an axiom from which to begin construction.

A typical example is the grammar {*A*,*B*, [,], (,)} and the two production rules [48]:

$$A \rightarrow AA$$

$$B \rightarrow A[B]AA(B)$$

The graftal is built by recursively feeding the axiom through the production rules. Each character of the input string is checked against the rule list to determine which character or string to replace it with in the output string. In this example, a '*A*' in the input string becomes '*AA*' in the output string. The axiom is applied in the same manner for '*B*':

1st Recursion: A[B]AA(B)

2nd Recursion: AA[A[B]AA(B)]AAAA(A[B]AA(B))

This string is drawn as an image where each symbol is assigned a graphical operation; the square brackets become a left branch and parentheses a right branch (Fig 2.18a).

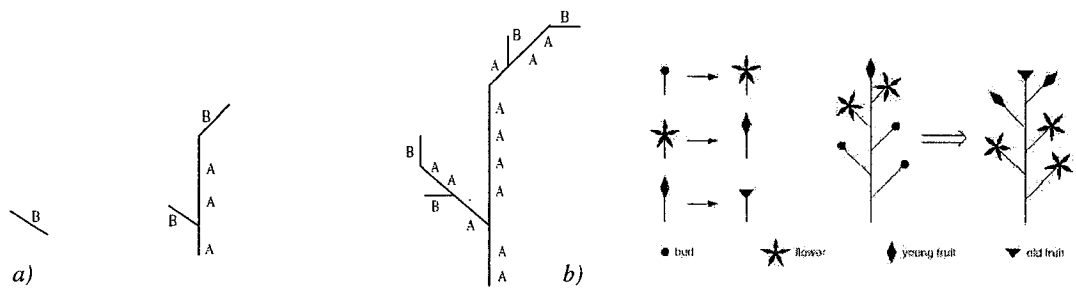


Figure 2. 14. (a) Tree representation of the first three words of the grammar (redrawn from [48]). (b) Representation of a plant (reproduced from [76])

Advantages

Fractals and graftals create surfaces using an implicit model that produces data when requested. They save on storage space and object construction time and allow a terse representation of a complex model. Graftals share with fractals the property that they offer infinite detail.

Disadvantages

Rendering can be difficult and slow.

Applications

Fractal models have been used to model natural objects that exhibit self-similarities: mountain landscapes with rivers [76], rocks, clouds and coastlines, planets and virtual universes [77], and in the architectural domain [79]. The plants models are the most spectacular examples

of *graftals* modeling. However, the method has been used in other applications as well, such as architecture (city and street modeling) [80]. Actually in any domain in which the objects being modelled exhibit sufficient regularity, there may be an opportunity to develop a grammar-based model.

2.4.2. Particle Systems

A *particle system* is defined as a collection of simple primitive objects (particles) that evolve over time [48, 51] and that are processed as a group in order to represent an object [81]. The evolution is determined by applying certain algorithms, usually in form of probabilistic rules to the particles [48, 51]. The characteristics of these objects, such as shape, size, position, color, and the lifetime of the particle itself, can be changed dynamically using a set of parameters. Stochastic processes that randomly select each particle's appearance and movement are constrained by these parameters. In general, each parameter specifies a range in which a particle's value must lie [78]. If these parameters of the particles are coordinated, the collection of particles can represent an object.

Advantages

Advantages over classical surface-oriented techniques are presented in [78, 82]. One advantage is that a particle is a much simpler primitive than a polygon, the simplest of the surface representations. Therefore, in the same amount of computation time one can process more of the basic primitives and produce a more complex image. Moreover, the model definition is procedural and stochastic. Therefore, obtaining a highly detailed model does not necessarily require a great deal of human design time as is often the case with existing surface-based systems. Because it is procedural, a particle system can adjust its level of detail to suit a specific set of viewing parameters. As with fractal surfaces, zooming in on a particle system can reveal more and more detail. Last, but not least, particle systems can be used to model dynamic objects which are more difficult to represent with surface-based modeling techniques.

Disadvantages

Particles of changing sizes can lead to performance penalties. Moreover, particle systems are hard to integrate seamlessly into a complex scene [81].

Applications

Several applications include modeling solid objects [53] (a deformable solid is modeled as a 3D array of particles that are held together by springs). When the object is subjected to external forces, the particles move and their positions approximate the shape of the solid object),

simulating fire [78], smoke, fog, explosions [78], fireworks [78], water flows, trees [82], grass [82], whirling galaxies, flocking behaviour of birds and other natural phenomena. Particles are also a method of animation. In this context, the main exploited idea is that particle systems are provided with control means for creating, moving, changing, and deleting particles during their lifetime.

2.5. Neural Networks in Object Modeling Context

2.5.1 Introduction

Neural networks are designed similarly to the biological neural pathways and perform the same basic task: processing input data, arranging this data into some sort of meaningful order, and then passing the information to the outputs.

The basic architecture of a neural network consists of a large number of simple processing elements, called neurons. Each neuron has an internal state, called activation function, which is a function of the inputs it receives. The neurons are connected to form a network. The way in which neurons are connected defines the network *topology*. Most neural networks have a layered topology consisting of an input layer (neurons in this layer receive external inputs from outside the network) and an output layer (neurons in this layer produce some of the outputs of the network). The inner layers of a network are referred to as "hidden layers" since they receive neither input from nor output information from outside of the network.

The neurons are capable of storing information, which determines their output for any given input. By convention this information is stored in the weights associated with each communication link. Neural networks can modify this information during the *learning* process. Learning consists in the adjustment or adaptation of the network weights or parameters over time, such as to permit the system to perform the task it has been designed for. There are three types of learning: supervised, unsupervised and reinforcement learning. In the supervised learning, training is accomplished by presenting a sequence of training vectors, each associated with a target output vector. An error is computed between the output and the target vector. This error provides an external feedback, which is used to adjust the network. In unsupervised learning a sequence of input vectors is provided, but no target vectors are specified. The net adjusts itself such that the most similar input vectors are assigned to the same output unit [87]. Reinforcement learning is an intermediate between supervised and unsupervised learning. Instead of providing total feedback (the error in supervised learning) or no feedback (as in unsupervised learning), a moderate feedback is obtained.

Used either as a standalone tool or as complement to classical modeling approaches, neural networks have been applied in several areas of computer graphics for data fusion [5, 6] and improvement of measured data using *a priori* knowledge [7, 8], projection, visualization and exploration of high-dimensional data [9, 10, 11], reconstruction of surfaces and shapes from scattered data, range data and shape from shading [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26] object representation and modeling [27, 28, 29, 30, 31, 32, 33, 34, 35, 36], modeling of isosurfaces [37] segmentation [30, 38], shape matching [39] and volume matching [40], object classification [15, 16, 41] and object motion [15,16, 42, 43, 44].

2.5.2. Data Fusion, Enhancement and Visualization

Often, in the process of object modeling, information from various types of sensors has to be verified, integrated and registered. In this context, neural networks have successfully been used for data fusion and enhancement. Lilienblum *et al.* [7,8] use an *associative memory* to smooth and correct the calculated 3D coordinates using *a priori* knowledge about a measured object. An associative memory can be seen as a one-layer neural network where all neurons function both as input and as output units. It stores mappings of specific input representations to specific output representations. The neural network proposed by Lilienblum *et al.* stores known shapes of objects and is able to interpolate between them to compute the correct shape. The storage of *a priori* knowledge is done online. A surface profile of the model object is constructed using CAD or using direct measurements. This model is superimposed with noise, thus changing its shape in small dimensions. The new obtained models using this procedure are used as training set for the network. When presented with the approximate 3D coordinates obtained by on-line measurement, the network prompts the results for the correct 3D coordinates. This model reduces the random and the systematic errors in the data set.

An IFOSART neural architecture becomes the core component of a refinement system that supplements the 3D construction model in [6]. An IFOSART is an incremental fully self-organizing simplified adaptive resonance theory ART neural network. As an ART network, it is able to dimension itself for data clustering, but is also able to contract to eliminate noise in the input patterns. It is self-organized and incremental since it dynamically and incrementally expands to incorporate the input data. It also enables supervised learning, by incorporating besides the input vector, a target vector in the learning procedure. The purpose of the system based on IFOSART in [6] is to verify and validate range information using the corresponding intensity data, and thus allow for the use of less accurate low-cost range scanners in the data acquisition stage. The refinement consists in iterative noise removal. Having as input data the

differences in intensity and range between surface patches centered about a point on the model object and the acquired data about the same point, the neural network learns the noise accrued in data acquisition. Because this process is done independently for each patch and overlapping areas are not considered, in a second stage, a similar refinement is done on all 3D points of the object obtained by integration of refined patches for all viewpoints. The final correction of the point is thus a combination of all the refinements over the set of viewpoints.

Mostafa *et al.* [5] integrate shape from shading with range data using a multilayer feedforward neural network trained using extended Kalman filter-based training.

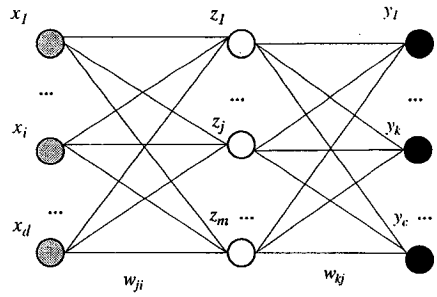


Figure 2. 15. A two-layer feedforward neural network

A feedforward network is a network in which the information only flows in a forward direction from the input neurons through the hidden neurons to the output neurons, without any connections back to previous neurons. A multilayer feedforward network (MLFF) is one in which the neuron outputs of one layer fed into the neuron inputs of the subsequent layer. The neurons in each layer sum the weighted input signals, apply the activation function (nonlinear activations functions are preferred) and pass the output to the next layer. They are trained in supervised mode. MLFF structures are known to be good function approximators and thus the integration proposed by Mostafa *et al.* is based on propagating the error difference between the two data sets by fitting a surface to that difference.

Another category of applications of neural networks is in the area of projection, visualization and exploration of multi-dimensional data. The most representative architectures in this domain are the self-organized architectures. A Kohonen self-organizing map (SOM) contains one layer of neurons, but two layers of connections. Each neuron has external connections to the inputs and is also laterally connected to its neighbours up to a certain distance. Computations are feedforward in the first layer of connections. Each neuron in this layer computes the weighted sum of the received inputs, hence each neuron produces a selective response to a particular set of inputs [92]. The second layer of connections is a feedback excitatory/ inhibitory network, with the purpose to reinforce the activation values of 'strong' neurons and decrease the activation of

'weak' ones [95], as depicted in Fig.2.16 for the case of a 1D output grid (the connections are no longer shown for the 2D grid to avoid burdening the figure). Unlike the connections in the input layer that are strictly feedforward, the ones in the connection layer are bidirectional, with the same weight in each direction as it depends on the distance on the grid. The training is unsupervised.

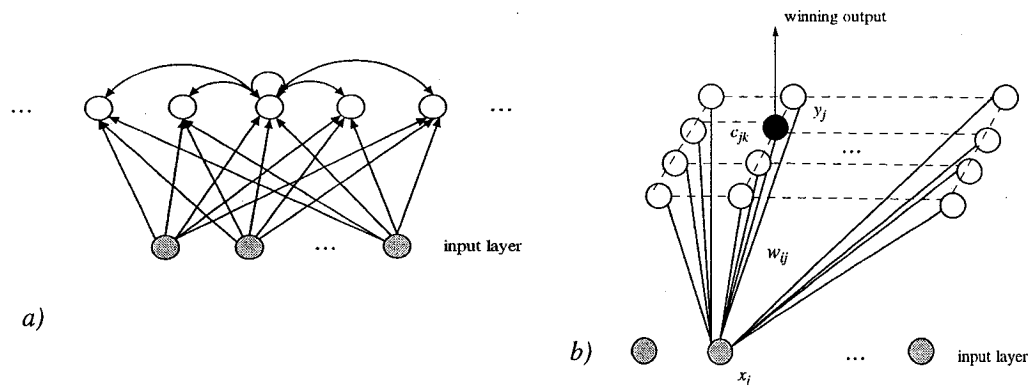


Figure 2. 16. (a) 1D grid Kohonen SOM (redrawn from [92]) , (b) 2D grid Kohonen SOM

The SOM has the potential to map a high-dimensional input space to a low-dimensional representation that preserves the topological ordering of the original inputs. When the items are mapped to those units on the map that have the closest reference vectors, nearby units will have similar data items mapped onto them [10]. Such an ordered display of the data items facilitates understanding of the structures in the data set. Moreover, the same ordered display can be used for illustrating the clustering density in different regions of the data space. The density of the reference vectors of an organized map will reflect the density of the input [10, 11]. A similar idea of transformation of data prior to use is proposed in [9]. The transformation entails the derivation of a new set of data defined by the distances of the original data from a given set of reference vectors. Both the transformation and the subsequent mapping are optimized simultaneously using an autoassociative neural network. The technique allows for the processing data to be viewed from various perspectives, while retaining the original structure but is effectively limited to data sets of relatively low dimension.

2.5.3. Representation and Reconstruction

Neural network architectures have been applied to represent data from various sources such as range and intensity data, structured light, shape from shading and images, and also for modeling of meshes and isosurfaces.

One of the frequently encountered solutions for the modeling of 3D objects is using a neural network that has as inputs the 3D Cartesian coordinates of a point and that learns a volumetric representation of an object. In Tseng [28,29] and Hwang [15,16], objects are represented parametrically by a continuous distance transform neural network, with the structure of a standard feedforward neural network. The architecture is trained using backpropagation, with the surface points of a reference object. The network maps any 3-D coordinate into a value that corresponds to the distance between the point and the nearest surface point of the object. Two regularization terms are used to remove the unbounded erroneous surfaces and to form a constant slope profile. After training, a mismatch between the reference object and an unknown object (given as range data) can be easily computed. When encountered with deformed objects, this mismatch information can be backpropagated through the network to iteratively determine the deformation in terms of affine transformations [28, 29]. The model can also be used to estimate motion of 3D objects [16].

A similar modeling approach is used by Kumazawa [32, 33] and Piperakis and Kumazawa [27]. A feedforward multilayer backpropagation network outputs 1 if the point is inside the object and 0 if outside. This choice is justified in [32, 33, 83, 84]. An object can be considered an "AND" of edge description nodes (planes or curved surface patches). These nodes are then connected finally by an "OR" operator to create the final object from patches. The representation is accurate and compact and affine transforms can easily be applied using these models [27]. The networks use pointclouds of synthetic and range data, but can be applied as well to images. Based on the same idea of model representation, Kumazawa [39] describes a shape matching method that transforms a template shape to fit target shapes included in a sequence of image frames and Kumazawa and Ohno [33] extend it to 3D. A 3D shape model is constructed from bitmap images for a target shape by deforming the 3D shape model such that each of its silhouettes from multiple viewpoints fit to a corresponding silhouette of the target shape.

The same feedforward network architecture with 3 inputs corresponding to 3D coordinates is used by Carcenac and Akan [37] to model the scalar field underlying an isosurface. The network is trained using conjugate gradient backpropagation with the control points of the isosurface and has as output the field value at the given point.

Other modeling approaches work in conjunction or exploit elements of classical modeling approaches. Ivrissimtzis *et al.* [35] transform polygonal models in a neural network model. A SOM network is employed as a mesh, with the nodes representing the vertices. The information stored in the node consists of the position of the vertex and a scalar value that measures the activity of the vertex during learning. The connections between nodes correspond to

the edges. The adjustment refers to the connectivity of the network. The algorithm works by sampling randomly a point cloud or an implicit surface and adjust accordingly a growing cell structure. It starts with a very simple network to which nodes are added incrementally. First a sample point is chosen from the target space and the winner of the network is found (the vertex that has the shortest distance from the sample point). The position of this vertex and its scalar value is updated together with that of its topological neighbours. After a number of iterations the most active vertices are split and the inactive ones are removed. The algorithm yields satisfactorily results for sharp features and concavities and the speed is virtually independent from the size of input data.

An alternative approach assumes the existence of a predefined mesh (that correctly describes the topology of an object), which is given *a priori* by the user [34]. A Kohonen SOM neural network, with nodes representing vertices, the weight vector consisting of the coordinates at each vertex and the input point set as the set of input vectors is employed to obtain the correct 3D coordinates such that the shape of the mesh approximates that of the input point set. The algorithm cannot reproduce well concave surfaces and thus a mesh optimization method is used to remove the problematic triangles (edge swapping).

A similar Kohonen SOM is proposed as a basis for parametrization in [17,18]. Each neuron contains geometrical information about the node coordinates and neighbourhood relationships. The connections between the neurons do not change during training, but the neurons get different geometrical values. As the boundary of the NN does not coincide with the boundary of projected points, two solutions are proposed to correct this problem. In the "boundary last" method, the boundary neurons are attracted towards the projected points outside the grid by increasing the weight of the sampled points. This method has the advantage of detecting the orientation of the grid without user intervention, but remains problematic for concave boundaries. With the "boundary first method", the boundary is trained separately from the whole grid. The method gives a better boundary than the previous method, especially for concave surfaces. The same architecture is used for reconstructing a single B-spline surface from a single range image [18]. The role of the SOM in this case is to construct a base surface that will be further improved until the surface approximation error satisfies a given tolerance.

Another flavour of the above-presented methods for mapping a mesh into a 3D neural network is proposed by Bona and Salvetti [40]. Their "Volume-Matcher 3D" registers a source volume (SV) into a target volume (TV). The matching of pairs of volumes is based on the individuation of corresponding voxels considering criteria such as distance, gray value difference, gradient angles, gray level average difference, normalized gradient distance and mean square

error. First, the one-pixel width closed lines with a constant gray value defining the boundaries of the uniform regions of interest are computed and a 3D mesh is built that approximates the surface of the structures in the examined scene. The mesh is then associated to a network topology assuming that its vertices are coincident with nodes of the network. A registration between SV and TV can be obtained by training the network computed for SV with a number of stimuli selected from TV.

An adaptive technique to reconstruct a smooth surface from Bezier patches is presented by Knopf and Kofman in [13]. A neural network that performs a weighted summation of Bernstein polynomial basis functions is employed. Surfaces are reconstructed by simultaneously updating networks that correspond to the separate patches. The smooth transition between adjacent patches is obtained in the network by introducing additional constraints for positional and tangential continuity in the learning algorithm. Each network determines the control points of a low-order Bezier patch that best fits, or approximates the data contained in the segmented region. The output of the network can be used directly as control points for constructing the Bezier surface patch. The Bernstein basis function network has been successfully implemented to approximate a dense set of three-dimensional data acquired by range sensors, but also to reconstruct the cross-sectional contours and surface geometry from CT images [14].

Neural networks are also used as a method fitting surfaces to data. In this context, a method for 3D object modeling based on an ensemble of multilayer SOMs is proposed by Chen *et al.* [31]. The model is mesh-like deflating model, where sampled points are assumed to be initially inside the membrane and apply attractive forces on the deformable model. Thus, the model contracts asymptotically towards the points until it fits them and takes the shape of the object encoded in the given points. The dynamic behaviour of the membrane is based on the mechanics of elastic materials and the motion of a single node is described in terms of static equilibrium of forces applied at that node. To address computational requirements, dynamic and neural SOM formulation are integrated for winner neuron calculation. The modeling technique is used for both 2D and 3D structured light data. However, it cannot be applied to models with holes and is restricted to object topologically equivalent to the sphere.

Another approach is used by Chella and Pirrone [30], who use a neural network system composed of a SOM and a feedforward NN to segment and model data with superquadrics. The range data or the shape from shading images are used as inputs in a SOM network that encodes the data distribution with a low number of units. Its outputs are segmented using k-means clustering. A feedforward network trained with fast backpropagation is then employed to model the implicit function of a superquadric. The model also takes into account the spatial pose of the

superquadric. Such a system can be a useful vision tool for a manipulator robot enabling it to create an internal symbolic representation of the spatial structure of the objects in its operating environment.

Other approaches for fitting surfaces to data make use of the radial basis function (RBF). A radial basis function has the topology of a three-layered feedforward neural network as in Fig. 2.15. Typical choices for activation functions are linear, cubic, Gaussian, multiquadric and thin-plate splines. For each neuron in the hidden layer, the Euclidean distance between its associated center and the input of the network is computed. Centers are defined points that are assumed to perform an adequate sampling of the input space. Finally, the neurons in the output layer compute a weighted sum of the hidden layer outputs. The architecture is trained in a supervised mode. Carr *et al.* [20, 21] show that scattered range data can be smoothed at low cost by fitting a radial basis function to the data and convolving with a smoothing kernel. A polyharmonic spline is used to model a signed-distance metric computed near the object's surface. Off-surface points are projected along estimates of the surface normal. Points on the surface are assigned a zero value and off point surfaces are assigned positive distances if outside the object and negative values if inside the object. A RBF is then fitted to the distance data.

A hybrid feedforward-RBF model is used for 3D shape reconstruction from shape from shading by Cho *et al.* [22, 23]. The hybrid model is justified on one hand by the fact that most real surfaces are neither lambertian nor ideally specular, and on the other hand by the fact that a feedforward network is able to better obtain the diffuse term, while the radial basis function with Gaussian activation can better approximate the specular term.

Neural networks are used as models and classifiers in medical applications by Hsu and Tseng [12] and Coppini *et al.* [41]. Hsu and Tseng propose a method to predict and create surface profiles of bone defects using a 3D orthogonal neural network. The coordinates of the skeletal positions around the boundary of a bone defects, computed from CT images using boundary detection, are the inputs into a single-hidden layer NN based on Legendre polynomials. The network is trained using gradient descent to learn the scattering characteristics of the bone. After the NN is trained, the mathematical model of the bone defect surface is generated, and pixel positions are derived. These pixels are used to obtain a CAD model of the defect by applying reconstruction algorithms.

In [41], the surface of the left ventricle (LV) is represented as a closed, thin, elastic surface and the data as a set of radial springs acting on it. Each spring joins the 3D boundary point to the surface. Based on the assumptions that the LV surface is usually regular and smooth, closed and it does not fold, an algorithm for automatic recovery of LV from echo views is

developed. The ventricle contour is extracted from the echo images using classical edge detection and a feed-forward neural network trained with backpropagation is used to classify the edge segments into LV-boundary or noise. The final outputs of the boundary detector are the spatial coordinates of all the edge-points corresponding to LV edge-segments. These points are converted to 3D points using trigonometric calculations and are fed as inputs into a surface-recovery NN network. The modeling of left-ventricle in terms of geometry and motion is also described in [28, 29].

Other applications of neural network are for motion emulation. Costa *et al.* [85] simulate human arm movements in order to check the ergonomic features of various environments. The system consists of two multilayer perceptrons with feedback, connected in cascade and trained with backpropagation through time. The first network has as inputs the 3D coordinates of the following point of the arm trajectory and it emulates the kinematics of the movement, while the second one has as inputs the outputs of the first one plus 6 angle coordinates and as output one of the 6 possible postures. The simulator is trained with hundreds of movements and is able to generate a pseudo-human movement for whichever trajectory within the hemisphere. Ishikawa *et al.* [44] propose a neural network solution for 3D facial expression estimation and generation from 2D images. The facial tissue model is implemented as a collection of nodes (mass, position, velocity, acceleration, net force) and spring structures (stiffness, length, pointers to the nodes they are connected to). A marker is attached on each feature point of the subject's face to measure and model a facial expression. The face is divided into three subareas, each giving an independent skin motion. A 4-layer backpropagation network, trained with several patterns of facial expression, converts the marker movements into muscle contraction forces for each of these subareas. Newton's laws of motion govern the response of the model to force. Good results are obtained for modeling anger, disgust, fear, happiness, sadness, surprise.

Finally, neural architectures have been employed to recover 3D surfaces from images. A framework for recovery of 3D surfaces of faces from single images, based on backpropagation neural networks, one per each particular face part is proposed by Nandy *et al.* [24]. Each face part is associated with intensity data and range data parameterized on the set of view directions, scales and illumination directions. The network learns the many-to-one mapping to recover the local 3D shape from the local image shading pattern. The reconstructed parts of range data which are defined to have rectangular fields of view in the input domain, are integrated in a smooth manner by minimizing the difference in heights of the overlapping pixels of neighbouring regions. A similar solution based on SOM is presented in [25] where 3D shape is recovered from multiple 2D images taken from different viewpoints.

Chapter 3

A Multilayer Feedforward Neural Network for 3D Objects Representation and Transformation

The chapter discusses implementation issues of a multilayer feedforward architecture for 3D object representation. It starts by giving a quick overview on neural networks' learning ability and the critical factors for a successful learning procedure. It then describes the proposed architecture, the data sets used for training, the training algorithm and the testing strategies. Finally it examines possible applications of the proposed architecture in the virtualized reality framework.

3.1. Introduction

As mentioned in the previous chapter, neural networks can modify the information stored in their weights during the *learning* process. Learning consists in the adjustment or adaptation of the network weights or parameters over time, such as to permit the system to perform the task it has been designed for. A successful learning process causes the weights to change and eventually to stabilize [86].

A *learning rule* describes "a learning process that modifies a specified neural network to incorporate new information" [86]. There are two standard ways to incorporate new information. In the online training approach, data is fed in the network one observation at a time [87]. Thus the network updates the appropriate weights after each single input (and target) vector. In the offline

(batch) training approach, the network updates the weights after each complete pass through the entire sequence of training data. All data is analyzed together, simultaneously in this case.

The successful end of a learning procedure is *generalization* - the ability of the network to produce reasonable outputs associated with new inputs. The learning and generalization capability depend not only on the topology and size of the network, learning types and rules, but also on several other interrelated factors including data preprocessing, choice of activation function, initialization of weights and biases, the way in which training data is presented, its type and number of training samples used.

Of all the quantities than can be modified prior to a neural network training phase, the *data preprocessing* has the greatest effect on the convergence of the learning algorithm. It helps condition the computations, such that they are not as susceptible to round-off error, overflow and underflow [86].

There are many choices for the activation function such as the identity function, the binary step function, the binary sigmoid, or the bipolar sigmoid [87]. However the most common one and the most widely used is the sigmoid.

Initialization of weights can also have a significant influence upon both the training procedure and the final solution. Usually an initialization with randomly chosen small values is preferred to avoid symmetries in the network [83]. It is also important to avoid choices of these weights that would make the activation function values or the corresponding derivatives zero [86]. On the other hand, too small values can lead to slow training.

Last but not least, the learning procedure and the generalization ability of the network is influenced by the training data. The training data set is the data set that is used to train a given network. The number of data used in the training procedure depends upon the size of the network model and the type of model being used. A model with lots of weights requires generally a lot of training data, while a model with too few weights compared with the size of data set generally trains very slowly. Normally one should choose the smallest network that trains well and performs satisfactorily on the test data.

There are many ways of testing the generalization ability of a network. An empirical way of measuring the generalization error is that of cross-validation. The idea is to split the available data into two: one part is used to estimate the relation between the input and output (training set), and the other part to evaluate the generalization (testing set) [89].

All these aspects will be discussed for the proposed multilayer feedforward architecture for 3D object representation.

3.2. Topology and Data Sets

Based on ideas from some previous work in the domain [32, 33], the proposed architecture for 3D object modeling is composed of two modules: the transformation module and the representation module, as shown in Fig. 3.1, both implemented using standard multilayer feedforward (MLFF) neural networks.

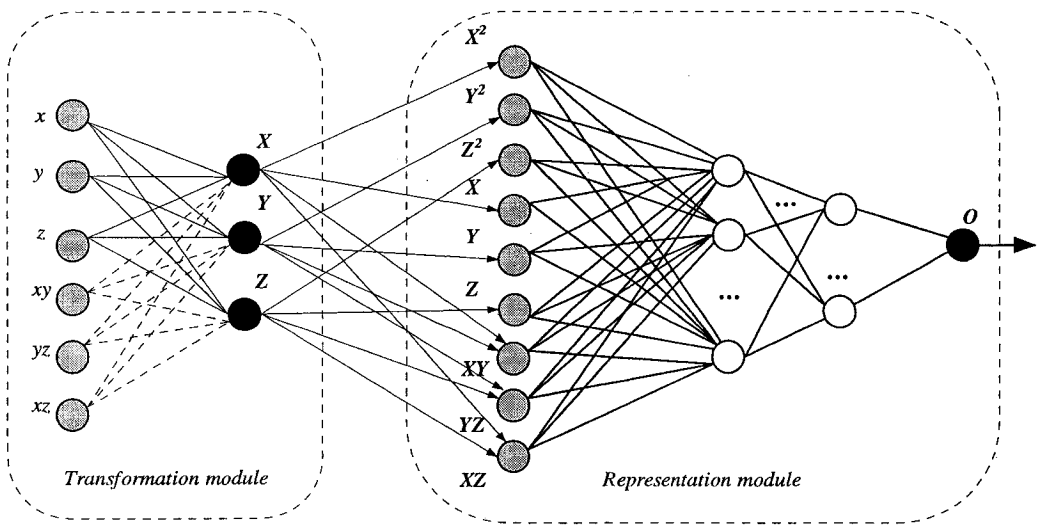


Figure 3. 1. 3D Neural Network Architecture

The choice for selecting a MLFF neural network is three-folded. First, MLFF neural networks are known to be able to represent high-dimensional non-linear functional mappings between inputs and outputs with the desired degree of accuracy, as long as the user is willing to the pay the computational time associated with it. Moreover, it has been proven that the class of MLFF neural networks with three layers of weights, as the one used in the representation module, is able to combine regions received as inputs into areas of arbitrary complexity [90], even if the corresponding areas are non-convex or disjoint [83]. This capability of MLFF networks combined with classical approaches of surface patch modeling is seen as a possible solution to achieve the goal of this thesis: 3D object modeling. Finally, MLFF neural networks being hardware implementable, they offer the possibility of further improvements in terms of computational time.

Range data pointclouds and also sets of synthetic data obtained with a 3D modeling tool are used to train the network. As in MLFF neural networks the support of hidden nodes is global, adjusting any weight will affect the output of the network for every input [11]. The training set

should therefore contain a relatively complete coverage of the input space otherwise the learnt network structure will not be of the correct form.

The number of training examples depends on the object's complexity and the quality of the data that represents it. A complex object described by just a few points will not give a good modeling performance. Thus a relatively large number of sampled points (range and synthetic) will be used for the initial pointcloud representations of complex objects, while a reduced number can be use for objects with moderate complexity.

The quality of training data used is another critical factor for the network's training. The quality refers first to the fact that the data should be preferably not noisy (ensured for synthetic data) or that the level of noise is low. This aspect is very important for the representation module. Moreover, data sets can exhibit large dynamic variances over one or more dimensions in the data. These large variances can often dominate more important, but smaller trends in the data. It has been found that problems with discrete binary values {0, 1} should be transformed into equivalent problems with bipolar values. This is because training problems are often exacerbated by 0 input values. These values are caused by wrong measurements and do not add any weight components because the corresponding input is 0, but the zero value also prevents the value from being corrected, as the error corresponding to that value is 0 as well [86]. Thus some data preprocessing needs to be performed on the data prior to using it.

3.2.1. The Representation Module

3.2.1.1. Purpose

The purpose of the *representation module* is to build a volumetric description of a 3D object. One network is used for each represented object. The network has as inputs the 3-D coordinates *X, Y, Z* of a point and their combinations (X^2, Y^2, Z^2, *XY, YZ, XZ*) and outputs a value proportional to the distance between the point given as input and the modeled object surface. The surface of an object is thus described by a set of zeros in the output response of a network, while the interior or exterior of an object are mapped to a negative or positive value, respectively. Therefore, the neural network model is in fact an implicit function for volumetric representations of objects.

3.2.1.2. Activation Functions

The sigmoid activation function is used for all the neurons of the representation module. This choice can be justified from two perspectives.

First, because it is bounded, monotonic, and differentiable, the sigmoid function is a standard for MLFF networks and their specific algorithm, backpropagation. The sigmoid function in its binary form is given by:

$$sig(x) = \frac{1}{(1+e^{-x\sigma})} \tag{3.1}$$

where the parameter σ controls the steepness of the curve. A small value corresponds to a gradual increase in the curve, while a large value corresponds to a steep increase, as shown in Fig. 3.2. Moreover, a sigmoidal hidden unit can approximate a linear hidden unit by making the values of weights and biases very small. In this case, the summed output will be itself small and will lie in the linear part of the sigmoid, near the origin. For $\sigma \to \infty$ the sigmoid becomes the hard-limiting step function. This can be obtained with very large values for weights and biases.

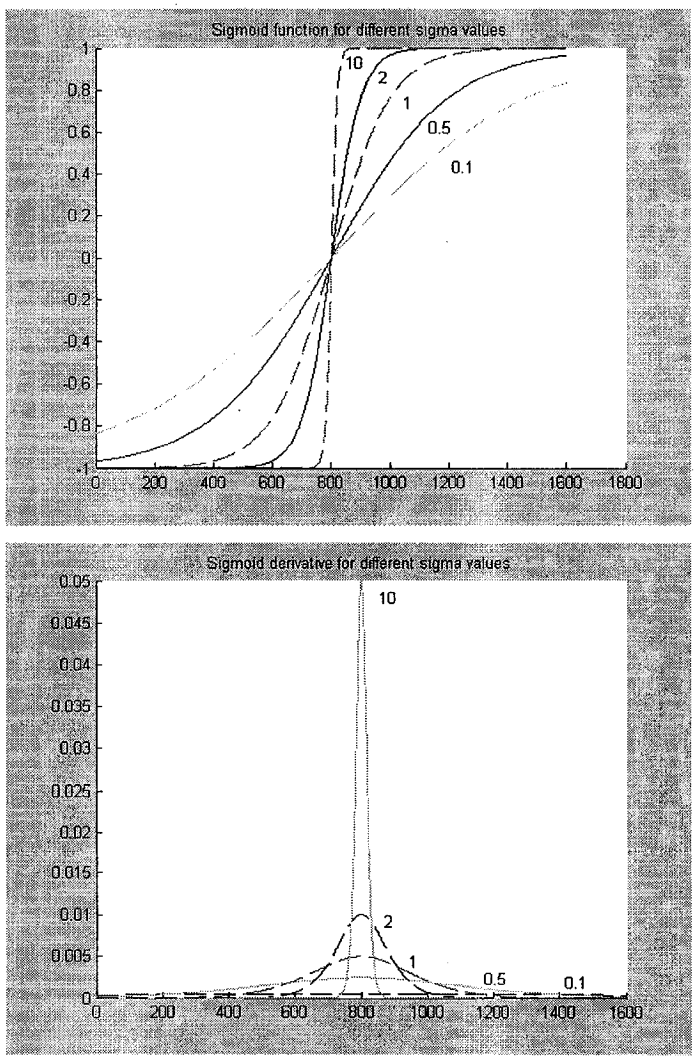


Figure 3. 2. Sigmoid functions and its derivatives for different values of σ

An important characteristic for almost all training algorithms is that the sigmoid is differentiable. Its derivative is given by:

$$sig'(x) = \sigma sig(x)[1 - sig(x)] \tag{3.2}$$

Second, complex object shapes can be represented using what Kumazawa [32, 33] calls "a tree of cascaded sigmoid functions" (the representation module presented in this thesis is an extension to the 3D case of the one proposed in [32]). It is in fact an analogy with classical modeling approach with patches on one side, and with the neural trees [84, 91] on the other side. A 3-layered architecture is used to model a 3D object. The input layer has the sole purpose of transmitting input patterns. In the first hidden layer each sigmoid function describes the shape of an object patch (plane or curved surface) that approximates the object surface. Using the sigmoid form given above, for a sufficiently large value of σ, the regions outside, or inside of a patch can be represented. The activation function will return a 1 on one side of the patch and 0 on the other side. For example, to represent points inside of spherical surface, a function like $sig(1-x^2-y^2-z^2, \sigma)$ can be used. The output will take 0 for (x, y, z) inside the spherical surface and 1 for outside. In the general case, a patch describing neuron will be represented as $sig(c_1+c_2 x+c_3 y+c_4 z+c_5 xy+c_6 yz+c_7 xz+c_8 x^2+c_9 y^2+c_{10} z^2, \sigma)$.

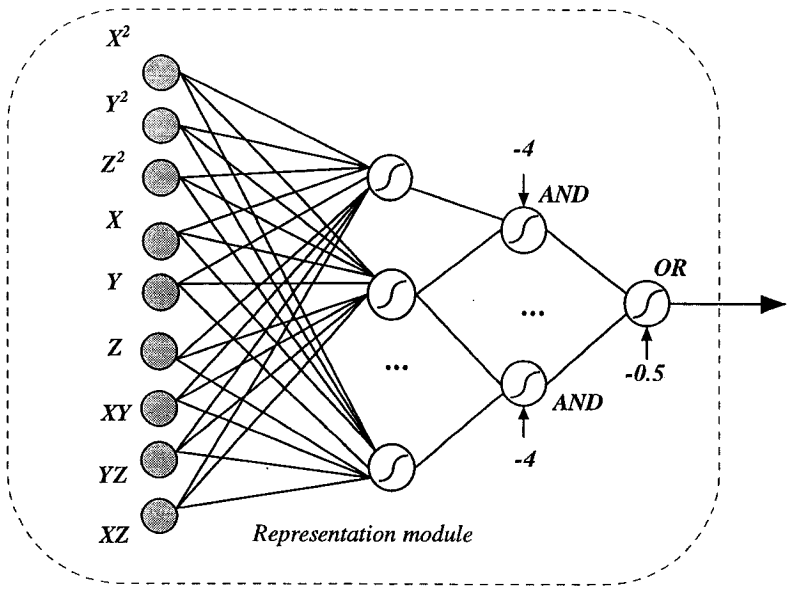


Figure 3. 3. Representation module with activation functions

Each sigmoid in the second hidden layer combines the patches represented in the first layer by an "AND" operator. These neurons will share the space in regions, inside or outside

given patches. The "AND" operator is implemented in such a way that it can describe plane and curved surfaces, either concave or convex. A minimum of 6 patches is considered, since this is the minimum number of patches that can describe a 3D volume of whatever complexity (concave or convex). To ensure that the "AND" operation is performed correctly using a sigmoid, a sufficiently large value for σ is needed ($\sigma = 5$ is used during training) – to create a quite steep increase and a bias of -4 is used (to direct the output into the correct value).

The outputs are then connected with an "OR" operator to obtain the final shape of the object. Again, to ensure that the "OR" operation is performed correctly by a sigmoidal function, a bias of –0.5 is used.

3.2.1.3. Working Modes

The representation module can be used in *generation mode* to obtain representations of simple objects whose equations are known [27]. The weights and biases are set such that the output neuron computes the equation that represents the desired object. In this case no hidden layers are needed.

For example, in order to model an ellipsoid whose equation is:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{x^2}{c^2}-1=0 \tag{3.3}$$

the X^2, Y^2, Z^2 inputs of the representation module will be used, with the weights $1/a^2$, $1/b^2$, $-1/c^2$, the bias –1. When tested for a given set of points in 3D space, such a network will reply with a positive value if the given point is outside the ellipsoid and with a negative value if the point is contained in the ellipsoid.

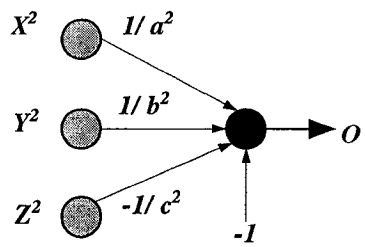


Figure 3. 4. MLFF neural network representation for an ellipsoid

The same architecture can be used in *training mode* to learn volumetric representations of objects given in form of pointclouds. Such data can be obtained from image acquisition systems (range data in this thesis) or using 3D modeling tools. In this case the user has to decide upon which inputs to use, the number of neurons in the hidden layer and the values for the training

parameters. Intuitively, an object with only flat surfaces will use X, Y, Z inputs and one with curved surface will use X^2, Y^2, Z^2 inputs and the combinations of X, Y and Z. When no *a priori* knowledge is available on the shape of the object, at least X, Y, Z and X^2, Y^2, Z^2 should be used in order to ensure that an object with variable complexity can be represented.

3.2.1.4. Data Preprocessing

As mentioned before, some preprocessing must be performed on data prior to its use. One technique for removing the variations in data sets and avoid the binary values {0,1} is normalization. Normalization removes redundant information from a data set, typically by compacting it on an interval. In this case, all the points of a given object (either real range data or synthetic data) are normalized in the [-1 1] interval, using a simple heuristic. For each input vector x, a maximum value for the components $m = max\ (x_i)$ is computed and all the values are divided by this value. The object modeling space is thus the 3D cube [-1 1, –1 1, -1 1].

3.2.1.5. Network size

The size of the network model depends on the number of data samples used to train it. A model with lots of weights and biases to determine generally requires a lot of training data, or else it will memorize well, but not generalize well. That is, it may train faster and do quite well reproducing desired training results, but it may give an unsatisfactory performance when any kind of testing data (similar but different from the data in the training set) is used. On the other hand, a model with too few weights compared with the size of data set may train very slowly and in fact, not train at all. Normally one chooses the smallest network that trains well and performs satisfactorily on the test data.

The number of weights and biases in a feedforward network is equal to the number of connections, as they present a fully connected structure. A simple way to compute this number is:

$$W = \sum_{l=1}^{L-1} W_l = (W_I + W_{BI}) \cdot W_H + (W_H + W_{BH}) \cdot W_o \tag{3.4}$$

where W_l is the number of connections between layer l and l+1, L is the total number of layers, W_I is the number of inputs, W_{BI} is the number of biases for the input layer, W_H is the number of neurons in the hidden layer, W_{BH} is the corresponding biases and W_O is the number of outputs.

One aspect the user should be careful of, is that it has been proven that the probability of error increases with the number of layers in the MLFF network. Thus the number of hidden layers is reduced to two in the case of the representation module.

3.2.1.6. Extrasurfaces

Using only the points of the given object for the training of the representation module is not appropriate, because the object surface is a set of zeros in the output of the neural model. In order to avoid the trivial solution and improve the training process, a mean must be found to append data such that not all the target set is 0. Extrasurface points are added in the interior and exterior of the object at an equal distance from the object surface along the surface normals [20, 27]. All the interior points will be assigned a negative value proportional to the distance along the normal to the surface and all the exterior ones a positive value proportional to the distance along the normal, as depicted in Fig. 3.5. Thus the entire object space is modeled.

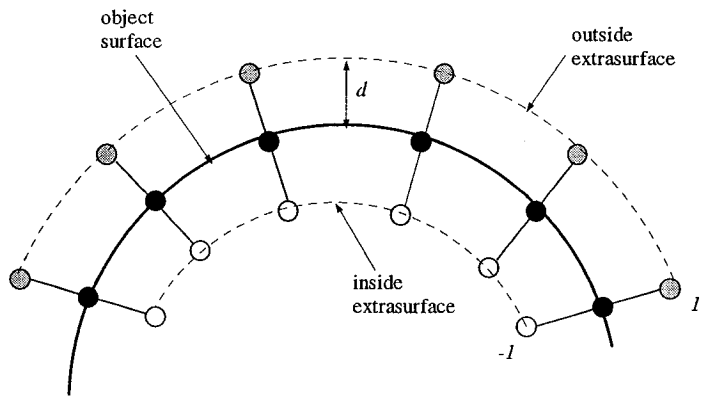


Figure 3. 5. Extrasurfaces

The algorithm for adding extrasurface points is quite simple [27]. Given a point $P(x_p, y_p, z_p)$ and the normal to the surface in that point denoted by $N(x_n, y_n, z_n)$, outside points are computed using the formula:

$$P_{em}(x_p + mdx_n, y_p + mdy_n, z_p + mdz_n) \tag{3.5}$$

where d is the distance between two extrasurfaces along the normal and m goes from 1 to the number of extrasurfaces. All these points will be assigned a positive value of magnitude $m*d$. Similarly, the inside points will be computed using:

$$P_{im}(x_p - mdx_n, y_p - mdy_n, z_p - mdz_n) \tag{3.6}$$

and will be assigned a value of magnitude $-m*d$.

The number of extrasurfaces to be used for an object depends much on the object complexity and the number of points that describe the object in its pointcloud representation.

Also regarding the number of extrasurfaces, it has been shown that the resulting models are better, if an inside extrasurface and the corresponding outside extrasurface are used together at a time.

The distance at which the extrasurfaces are built is another critical factor for the learning. A too small value for this distance tends to create rounded edges, while a too large value makes it impossible for the network to learn the given object. The distance must be chosen such that the extrasurfaces do not touch the object surface and do not touch each other. This is especially important for objects with holes and/or concave or convex surfaces. If a normal in a point intersects another surface of the object, the model will be distorted in the region around that point.

3.2.2. The Transformation Module

3.2.2.1. Purpose

The *transformation module* implements transformations such as rotation, translation, scaling and deformations such as bending, tapering, and twisting. It receives as inputs the 3-D coordinates *x, y, z* of a point and their combinations and returns the 3-D coordinates of the corresponding transformed point. The combinations *xy, yz, xz* are used to model the tapering deformation.

3.2.2.2. Activation Functions

The network has no hidden layer and the 3 outputs have linear activation functions. A linear transfer function is chosen, such that it simply passes the result to the output. The output value is only altered by the biases' values that will be used in this case to model the translation components. The scaling and rotation will be encoded in the weight matrix.

3.2.2.3. Working Modes

Knowing the equations for each of the transformations, the network can be used in *generation mode* to compute the new coordinates *X, Y, Z* of the transformed point just by setting the weights of the neural network. If this procedure is repeated for each point of an object, the transformed object can be easily obtained by plotting all the transformed points.

For the translation, rotation and scaling, the weight matrix of the network will be nothing else than the generalized matrix in TRPY convention, where the translational components will pass as biases. The affine transformation network will have the structure presented in Fig. 3.6. Only the *x, y, z* inputs are needed.

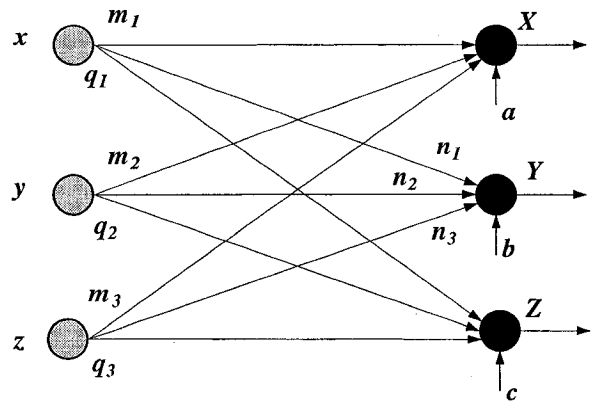


Figure 3. 6. Affine transformation neural network

The weight matrix is defined by:

$$
\begin{aligned}
m_1 &= a_1 \cos\theta \cos\varphi \\
m_2 &= a_1(\cos\theta \sin\varphi \sin\psi - \sin\theta \cos\psi) \\
m_3 &= a_1(\cos\theta \sin\varphi \cos\psi + \sin\theta \sin\psi) \\
n_1 &= a_2 \sin\theta \cos\varphi \\
n_2 &= a_2(\sin\theta \sin\varphi \sin\psi + \cos\theta \cos\psi) \\
n_3 &= a_2(\sin\theta \sin\varphi \cos\psi - \cos\theta \sin\psi) \\
q_1 &= -a_3 \sin\varphi \\
q_2 &= a_3 \cos\varphi \sin\psi \\
q_3 &= a_3 \cos\varphi \cos\psi
\end{aligned}
\qquad (3.7)
$$

where θ defines the rotation around z, φ the rotation around y, ψ the rotation around x, $a = x_T$, $b = y_T$, $c = z_T$ describe the translation and a_1, a_2, a_3 define the scaling factor along x, y, z respectively.

The same idea is exploited to model tapering, bending and twisting. Knowing the equations, the network weights will be set such that the desired deformation is performed. For example, to model generalized tapering, the equation of tapering as presented in section 2.3.4 are used. The global tapering along x is given by:

$$
\begin{aligned}
X &= x \\
Y &= f_y(x)y \\
Z &= f_z(x)z
\end{aligned}
$$

where X, Y, Z are the coordinates of the points in the deformed solid, x, y, z the coordinates of points in the original solid and $f_y(x)$ and $f_z(x)$ are the tapering functions along y and z axes. The

tapering functions are piecewise linear functions which decrease as x increases [65]. In this context, $f_y(x)$ and $f_z(x)$ are chosen to be:

$$f_y(x) = \frac{k_1}{a_1} x + 1, \quad k_1 \in [-1, 1]$$

$$f_z(x) = \frac{k_2}{a_1} x + 1, \quad k_2 \in [-1, 1]$$

where k_1, k_2 are the tapering parameters that control the decrease of the functions and a_1 is the scaling in the x direction [66]. Replacing in the tapering equation we have:

$$X = x$$

$$Y = \frac{k_1}{a_1} xy + y$$

$$Z = \frac{k_2}{a_1} xz + z$$

In the same way for the tapering along y and z:

$$X = \frac{k_{11}}{a_2} xy + x \qquad X = \frac{k_{12}}{a_3} xz + z$$

$$Y = y \qquad Y = \frac{k_{22}}{a_3} yz + z$$

$$Z = \frac{k_{21}}{a_2} yz + z \qquad Z = z \tag{3.8}$$

All these are then combined in a single matrix $W_{tapering}$.

A network with the structure in Fig. 3.7 will be thus used,

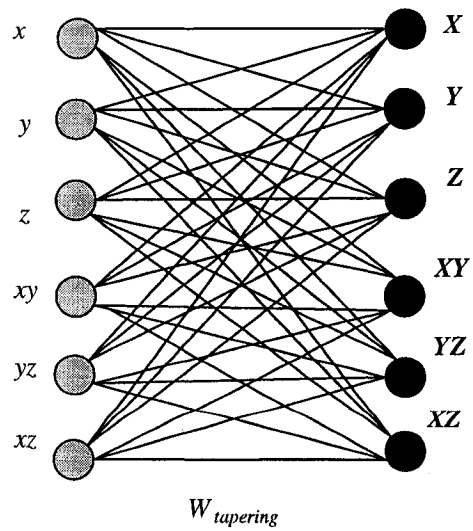


Figure 3. 7. Tapering neural network

where:

$$\begin{bmatrix} X \\ Y \\ Z \\ XY \\ YZ \\ XZ \end{bmatrix} = W_{tapering} \begin{bmatrix} x \\ y \\ z \\ xy \\ yz \\ xz \end{bmatrix}$$

$$W_{tapering} = \begin{bmatrix} 1 & 0 & 0 & k_{11}/a_2 & 0 & k_{12}/a_3 \\ 0 & 1 & 0 & k_1/a_1 & k_{22}/a_3 & 0 \\ 0 & 0 & 1 & 0 & k_{21}/a_2 & k_2/a_1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

k_1, k_2, k_{11}, k_{12}, k_{21}, $k_{22} \in$ [-1,1] are the coefficients of the tapering functions along x, y and z axes with $k_{11} = k_{12} = k_{21} = k_{22} = 0$ for tapering along x, $k_1 = k_2 = k_{12} = k_{22} = 0$ for tapering along y, and $k_1 = k_2 = k_{11} = k_{21} = 0$ for tapering along z.

Twisting and bending can be obtained in a similar manner. For twisting along x, the function $f(x)$ from eq (2.22), section 2.3.4 is chosen to be, for example:

$$f(x) = k\frac{\pi}{a_1}x, \quad a_1 \in [-1, 1]$$

and in similar manner is proceeded for y and z.

Up to now, the network is only used to compute the coordinates of points knowing the corresponding transformations. However, when provided with corresponding points from a given reference model as inputs and those of a transformed model as targets, the transformation module can learn and store in its weights information on how these points have been transformed and/or deformed. This information will be used for object classification and motion estimation, as shown in sections 3.4.4 and 3.4.5.

3.2.2.4. Data Preprocessing

As in the case of the representation module, a normalization operation will be performed on all the points of a given object (either real range data or synthetic data) such that they are scaled in the [-1 1] interval. Due to the fact that the module is used as complement to the representation module, and thus it is more a passive component, no other supplementary data will be used in training mode.

3.3. Training and Testing

3.3.1. Training Algorithm

The training of both networks representing the two modules of the proposed neural architecture is done in a supervised manner. *Supervised learning* is the process in which the external input data and the corresponding target data are provided and the network adjusts itself in some way such that, for a given input, it will produce the desired target. This can be done by determining the network output for a given input, comparing this output to the corresponding target and computing the error between the output and target. This error is used to provide the external feedback needed to adjust the network, as depicted in Fig.3.8.

As this process suggests, the training is based on the minimization of an error that is backpropagated through the network. For this reason, the used algorithm is backpropagation. The MLFF structure in Fig. 3.9 is used to explain the principles of the backpropagation algorithm in its online version. For the offline training approach, the only difference is that the update of the appropriate weights is done after each complete pass through the entire sequence of training data.

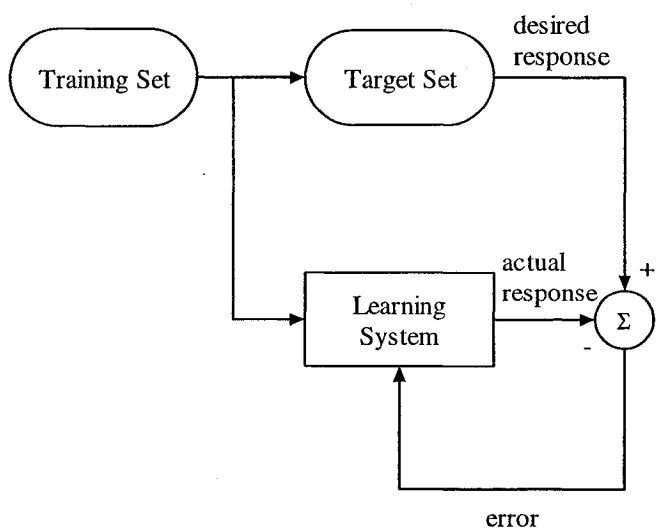


Figure 3. 8. Block diagram of supervised learning (adapted from [92])

There are d input units, m hidden neurons in the hidden layer and c outputs. w_{ji} denotes the weights associated with a connection from the input layer to the hidden layer (including the biases), where $i = 0,\ldots, d$, $j = 0,\ldots, m$ (start with 0 because they also include the biases), w_{kj} denotes weights associated with a connection from the hidden to the output layer (including the biases), where $k = 0,\ldots,c$, t_k represents the vector of target responses and f (see eq. (3.11), on the

next page) is a generic activation function, which is differentiable (the sigmoid in the context of this thesis).

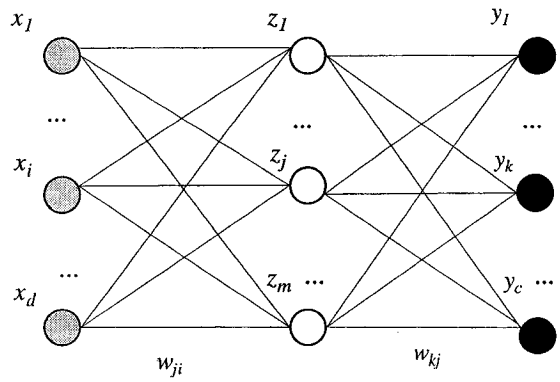


Figure 3. 9. MLFF architecture

The training of a network by backpropagation involves four stages stages: the initialization of weights and biases, the forward-propagation of input pattern, the calculation and backpropagation of the associated error and the adjustment of the weights.

3.3.1.1. Initialization of Weights and Biases

Initialization of weights has a significant influence on both the training procedure and the final solution. Usually an initialization with randomly chosen small values is preferred to avoid symmetries in the network [83]. It is important to avoid choices of these weights that would make the activation function values or the corresponding derivatives zero [86]. On the other hand, too small values can lead to slow training. A common procedure is to initialize the weights to random values between –0.5 and 0.5 or -1 and 1. However, in order to improve the learning time, some modifications are made to this random procedure.

For example, the Nguyen-Widrow method, used in this thesis, generates initial weight and bias values for a layer, such that the active regions of the input layer's neurons will be distributed roughly even over the input space. While the weight in the hidden layers are updated as in the random procedure with values between –0.5 and 0.5, the input units will be initialized such that the ability of the hidden unit to learn is improving. For each input pattern, the weights and biases will be chosen to push the output in the range in which the hidden neuron will learn "most readily" [87]. Given n input units and p hidden units, the weight vector is reinitialized for each step j ($j = 1, 2, \ldots, p$) with:

$$w_{ji} = \frac{\beta \tilde{w}_{ji}}{\|\tilde{w}_j\|}, \quad \beta = 0.7p^{1/n}, \quad \|\tilde{w}_j\| = \sqrt{w_{1j}^2 + w_{2j}^2 + \ldots w_{nj}^2}$$

(3.10)

where $\tilde{w}_{ji}$ is the weight vector from input units randomly initialized with values in –0.5 and 0.5. The same initialization is done for the biases.

3.4.1.2. Forward Propagation

After the initialization of weights and biases, the input patterns are forward propagated in the network, according to the following steps:

1. Each input unit receives the corresponding input signal x_i and broadcasts it to all units in the layer above (next layer from left to right).
2. Each hidden unit z_j ($j = 0,\ldots,m$) sums the weighted input signals, applies the activation function f (the sigmoid in the context of this thesis):

$$a_j = \sum_{i=0}^{d} w_{ji} x_i, \quad z_j = f(a_j) \tag{3.11}$$

   and sends the signal to the units in the layer above.
3. Each output unit y_k ($k = 0,\ldots,c$) sums the weighted inputs and applies the activation function f:

$$a_k = \sum_{j=0}^{m} w_{kj} z_j, \quad y_k = f(a_k) \tag{3.12}$$

3.3.1.3. Error Computation and Backpropagation

4. Each output y_k receives a target pattern t_k corresponding to the input pattern and compute its error information term (the mean square error is considered in this case, see 3.3.1.4):

$$\delta_k = (t_k - y_k) f'(a_k) \tag{3.13}$$

   and propagates it backwards. It also computes the weight and bias correction term:

$$\Delta w_{kj} = \eta \delta_k z_j \tag{3.14}$$

   where η is a fixed learning rate.
5. Each hidden unit z_j sums the δ_k received, computes its error information term:

$$\delta_j = f'(a_j)\sum_{k=1}^{c}\delta_k w_{kj} \tag{3.15}$$

and the correction term:

$$\Delta w_{ji} = \eta\delta_j x_i \tag{3.16}$$

Eq. (3.14) and (3.16) are the ones used in the case of simple gradient descent backpropagation. The algorithm modifies the weights in the direction of the fastest decrease in error (see 3.3.1.4, Derivation of learning rules).

3.3.1.4. Updating

6. After all training examples, each output unit y_k updates its weights and biases:

$$w_{kj} = w_{kj} + \Delta w_{kj} \tag{3.17}$$

7. Each hidden unit updates its weights and biases:

$$w_{ji} = w_{ji} + \Delta w_{ji} \tag{3.18}$$

The algorithm is repeated from step 1 until there are no more input patterns or some stopping conditions are true. The stopping conditions can be one of the following: the number of iterations through the algorithm is reached, the maximum amount of time is exceeded, or the performance function has reached the goal or fell below a given threshold.

Derivation of Learning Rules

As it was mentioned before, the training is based on the minimization of an error that is backpropagated through a network. Supposing that the error function can be written as a sum of errors over all presented patterns and is a differentiable function of the network outputs, the goal is to compute the derivatives of this function with respect to weights and biases.

In case the mean square error is used as the performance function, the error can be written as:

$$E = \frac{1}{2}\sum_{k=1}^{c}(t_k - y_k)^2 \tag{3.19}$$

and its first derivative is given by:

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{2}\cdot 2\cdot(t_k - y_k)\cdot\frac{\partial y_k}{\partial w_{jk}} = -(t_k - y_k)\frac{\partial}{\partial w_{jk}} f(a_k) = -(t_k - y_k) f'(a_k)\frac{\partial a_k}{\partial w_{jk}} =$$
$$= -(t_k - y_k) f'(a_k) z_j$$

Using the notation:

$$\delta_k = (t_k - y_k) f'(a_k) \tag{3.20}$$

it can be concluded that in order to modify the weights in the direction of the fastest decrease in error, an update as in Eq. (3.21) must be made:

$$\Delta w_{jk} = -\eta\frac{\partial E}{\partial w_{jk}} = \eta\delta_k z_j \tag{3.21}$$

Eq. (3.21) justifies the choice for the update function in eq. (3.14).

In the same manner, for the weights on the connections to a hidden unit, we have:

$$\frac{\partial E}{\partial w_{ji}} = -\sum_{k=1}^{c}(t_k - y_k)\frac{\partial y_k}{\partial w_{ji}} = \sum_{k=0}^{c}(t_k - y_k) f'(a_k)\frac{\partial a_k}{\partial w_{ji}} = -\sum_{k=1}^{c}\delta_k\frac{\partial a_k}{\partial w_{ji}} = -\sum_{k=0}^{c}\delta_k w_{kj}\frac{\partial z_j}{\partial w_{ji}} =$$
$$= -\sum_{k=1}^{c}\delta_k w_{jk} f'(a_j) x_i$$

Using the notation:

$$\delta_j = \sum_{k=0}^{c}\delta_k w_{kj} f'(a_j) \tag{3.22}$$

the update will be:

$$\Delta w_{ji} = \eta\delta_j x_i \tag{3.23}$$

as in eq. (3.16).

3.3.1.5 Optimization Methods

Although backpropagation reduces the complexity $O(W^2)$ that would be required to compute all the error functions and forward-propagate them to $O(W)$, the algorithm can still be very time consuming. This is mainly due to the fixed learning rate η. Moreover, the algorithm in its standard version (simple gradient descent) modifies weights in the direction of the fastest decrease in error. In general, this does not move the weights towards an optimal direction. Fig.3.10a illustrates the evolution of the local negative gradient vector along a zigzag path, progressing very slowly towards a minimum of error. The ellipses are contours of constant error, such that "the error surface has the form of a long valley "[83].

A variety of methods have been proposed to improve the capabilities of backpropagation. Different algorithms involve different choices for the updating vector Δw. A very simple technique is to add a momentum term to the gradient descent formula, with the effect of adding inertia to the motion in the weight space and to smooth the oscillations [83], as seen in Fig. 3.10b.

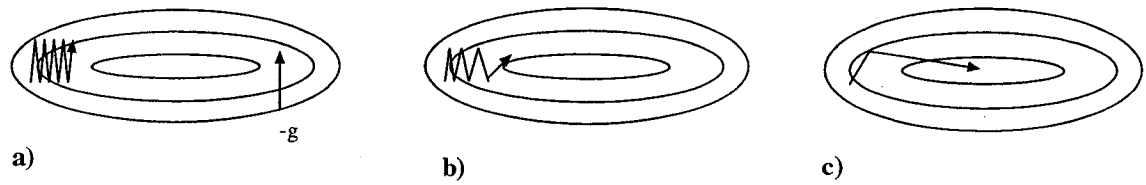


Figure 3. 10. Gradient algorithms (a) Fixed-step gradient descent, b) Gradient descent with momentum, c) Conjugate Gradient Descent) – redrawn from [83])

Although it leads to significant improvements in performance, the user has now to adjust not only the learning rate η, but also the momentum term, which is not an easy task. Line search algorithms have then been proposed to minimize the error along the search direction. In this case however, the gradient at the new minimum is orthogonal to the previous search direction and choosing successive search directions to be the local negative gradient direction can lead to the same problem as illustrated in Fig. 3.10a. Thus more sophisticated methods have been proposed that compute an approximation of the second derivative (Hessian) of the error function. The algorithms in this category are considered the fastest and are known to require the fewest number of iterations [86]. They are based on the assumption that a reasonably smooth function in N dimensions can be approximated by a quadratic function over a small enough region in the vicinity of an optimal point. They are usually combined with search algorithms to improve the step size. The scaled conjugate gradient algorithm used in this thesis belongs to this category.

3.3.1.6. Scaled Conjugate Gradient Backpropagation

The scaled conjugate gradient backpropagation is an improved conjugate gradient algorithm. This section discusses the basic principles of conjugate algorithms and shows the improvements proposed by Moller for the scaled version [93].

Conjugate Gradient Algorithm

For each iteration of the algorithm, a quadric function E_q is used to approximate the global error E in the current point w of the weight space.

$$E_q(w) = E(w) + g^T w + \frac{1}{2} w^T g^2 w, \quad g = g(w) = E'(w), \quad g^2 = g^2(w) = E''(w) \tag{3.24}$$

where g denotes the gradient and g^2 the Hessian of the error function. In order to minimize eq. (3.24), the critical points have to be first found. They can be obtained as the solution of the linear system defined by the equation:

$$E_q'(w) = g^2 w + g = 0 \tag{3.25}$$

If a conjugate system with respect to the Hessian is used:

$$p_i^T g^2 p_j = 0, \quad i \neq j, \quad i = 1,..,N \tag{3.26}$$

it can be shown that the vectors $p_1,\ldots, p_N$ are linearly independent (if the Hessian is positive definite) and the step from a starting point w_1 of the weight space to a critical point w_* is a linear combination of them.

$$w_* - w_1 = \sum_{i=1}^{N} \alpha_i p_i \tag{3.27}$$

Eq. (3.27) can also be also generalized in an iterative manner as:

$$w_{j+1} = w_j + \alpha_j p_j \tag{3.28}$$

In order to determine α_j, eq. (3.27) is multiplied by $p_j^T g^2$:

$$p_j^T g^2 w_* - p_j^T g^2 w = \alpha_j p_j^T g^2 p_j \tag{3.29}$$

and replacing $g^2 w_* = -g$ from eq. (3.25):

$$\alpha_j = \frac{-p_j^T(-g - g^2 w_1)}{p_j^T g^2 p_j} = \frac{-p_j^T g_j}{p_j^T g^2 p_j} \tag{3.30}$$

where $g_j = E_q'(w_j)$.

It results that using eq. (3.27) and (3.30), w_* can be estimated completely in N steps. α_j defines the step length along the search direction p_j. The search direction is determined by the coefficient β_j. The directions are chosen is such a way that they are not interfering (conjugate). This is achieved by selecting the first direction to be the negative gradient $-g$ and then choosing each successive direction to be the linear combination of the current gradient and the previous direction.

$$p_{j+1} = -g_{j+1} + \beta_j p_j \tag{3.31}$$

Imposing the conjugancy condition (3.26), we have:

$$\beta_j = \frac{g_{j+1}^T g^2 p_j}{p_j^T g^2 p_j} \tag{3.32}$$

But from (3.25), and applying eq. (3.27):

$$g_{j+1} - g_j = g^2(w_{j+1} - w_j) = \alpha_j g^2 p_j \tag{3.33}$$

from where

$$g^2 = \frac{g_{j+1} - g_j}{\alpha_j p_j} \tag{3.34}$$

Replacing (3.34) in (3.32), β_j becomes:

$$\beta_j = \frac{g_{j+1}^T (g_{j+1} - g_j)}{p_j^T (g_{j+1} - g_j)} \tag{3.35}$$

Eq. (3.35) is also known as the Hestenes-Stiefel expression.

Scaled Conjugate Gradient Improvements

The computation of the Hessian matrix is computationally expensive, therefore the first step in the series of 3-step improvement procedure is to use an approximation to compute the product of the Hessian with a vector.

$$g^2(w) p_j \approx \frac{g(w + \sigma p_j) - g(w)}{\sigma}, \quad 0 < \sigma << 1 \tag{3.36}$$

The second improvement refers to the fact that the above-mentioned conjugate algorithm works correctly only if the Hessian is positive definite (see eq. (3.26)). The idea to solve this problem is to add a scalar λ to ensure that the Hessian will be positive definite. Thus eq. (3.30) becomes:

$$\alpha_j = \frac{-p_j^T g_j}{p_j^T g^2 p_j + \lambda_j |p_j|^2} \tag{3.37}$$

where $|p_j|$ denotes the length of p_j.

Let $\delta_j = p_j^T g^2 p_j + \lambda_j |p_j|^2$. If $\delta \leq 0$, λ_j is raised by $\bar{\lambda}_j$ and the new $\bar{\delta}_j$ is computed as:

$$\bar{\delta}_j = \delta_j + (\bar{\lambda}_j - \lambda_j)|p_j|^2 \tag{3.38}$$

Moller [93] chooses

$$\bar{\lambda}_j = 2\left(\lambda_j - \frac{\delta_j}{|p_j|^2}\right) \tag{3.39}$$

Substituting (3.39) in (3.38):

$$\bar{\delta}_j = \delta_j + \lambda_j |p_j|^2 = -p_j^T g^2 p_j \tag{3.40}$$

which is now positive.

However λ_j scales the Hessian matrix even if the Hessian is already positive definite. Thus the third improvement refers to a mechanism of adjusting the value of λ_j in accordance. A measure Δ_j is introduced to compute how well E_q' approximates $E(w+\alpha_j p_j)$. The closer the value of Δ_j is to 1, the better the approximation is.

$$\Delta_j = \frac{E(w_j) - E(w_j + \alpha_j p_j)}{E(w_j) - E_q(\alpha_j p_j)} = \frac{2\delta_j (E(w_j) - E(w_j + \alpha_j p_j))}{\alpha_j p_j^T g_j} \tag{3.41}$$

λ_j is then adjusted as follows:

$$If\ \Delta_j > 0.75,\quad \lambda_{j+1} = \frac{\lambda_j}{2};\quad if\ \Delta_j < 0.25,\quad \lambda_{j+1} = 4\lambda_j \tag{3.42}$$

The value of λ_j is thus decreased if the approximation is quite good, or increased if the approximation is bad.

Thus, in each step of the algorithm the second order information in the denominator of eq. (3.30) is computed using the approximation (3.36). $\bar{\lambda}_j$ and $\bar{\delta}_j$ are then obtained using (eq. (3.38) and (3.39)). If $\bar{\delta}_j \leq 0$, the Hessian is made positive according to eq. (3.37) and the step length is obtained. Then the comparison parameter Δ_j is calculated and adjusted according to eq. (3.42). The algorithm goes back to computing the second order information for the next step j+1 if $g_{j+1} \neq 0$ or else returns w_{j+1} as the desired minimum. Because the error function E is non-quadric, it is possible that the algorithm does not converge after N steps. In this case, the algorithm is restarted.

3.3.2 Testing Procedure

The most important part of any learning system design procedure is the verification and validation phase, in which the user assesses how well the network has learnt the training data (memorization) and how successfully is it able to generalize to patterns similar to those provided during the learning procedure. The memorization capability can be tested by comparing the target values against the outputs of the network for the same data set used during the learning.

There are many ways of testing the generalization ability of a network. For the MLFF neural network model proposed, the testing is performed by evaluating it for points in the 3D object space [-1 1, -1 1, -1 1]. This can be done either in an orderly fashion, by dividing the [–1 1] interval in a given number of steps or using random generated points in this interval. To visually assess the results, only those points that belong to the interior of the object (with values less or equal to 0.1 – considering a 0.1 tolerance) are plotted.

In order to estimate the generalization error, *cross-validation* is used. The network is only trained using the training data set to determine the weights and biases, and a test data set composed of additional samples that were not used in training is selected. Each input pattern from the test set is presented to the network and the output is computed. Then the output is compared with corresponding target in the test set to determine each error. These errors are combined to produce the overall error for the given test set by using the same error measure that was used when the network was trained. This procedure is repeated for all data sets. If each overall error is small enough, then the network generalizes well [86].

The results of training and testing the MLFF neural network are the subject of section 5.1.1 (representation module) and 5.1.2 (transformation module) in chapter 5.

3. 4. Applications

The model given by the representation MLFF neural network has potential for applications in many areas of virtualized environments. The fact that it combines volumetric and implicit characteristics makes it an ideal candidate for modeling of object morphs and set operations and for detecting object collisions. Moreover, cascaded with the transformation module, it can be used for more complex operation such as object classification and recognition and object motion estimation.

3.4.1. Object Morphing

Taking the representations of two 3D objects (reference and target), 3D object morphing creates new objects that, viewed in order, smoothly change the first object into the second one. In order to perform object morphing, two assumptions have to be satisfied. First, the reference and target objects must be aligned (if not, the transformation module can be used as in section 3.4.4 to recuperate the displacement information). Second, a 3D neural model must exist for each of the objects. The two neural networks representing the two objects will store in their weights the volumetric description of each object. The objects will be represented using the same architecture (to ensure the weight matrix has the same size).

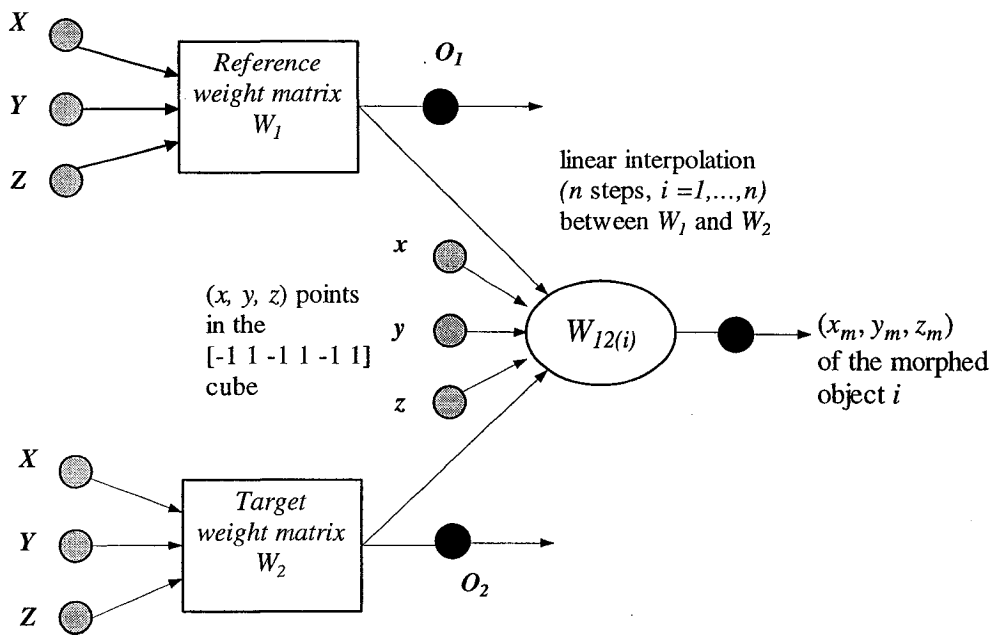


Figure 3. 11. Block diagram of object morphing

Given these, the morphing is performed by modifying the weights of the reference model to match the weights of the target model with a linear interpolation. If the reference model weight vector is p and the target weight vector is t, the weight vectors m of the corresponding morphed objects are obtained using the equation:

$$m = p + i * \frac{t - p}{n}, \quad i = 1,...,n \tag{3.43}$$

where n is the number of steps used in morphing to transform the reference object into to the target object. As the weights for each morphed object are known, the morphed object models are visually built using the representation module in generation mode. Points in the [-1 1 -1 1 -1 1]

cube are used as inputs, and only those whose output is smaller than 0 (meaning the points are inside the object) are plotted. The results are presented in section 5.1.3.1 of chapter 5.

3.4.2. Set Operations

As the neural network model of a 3D object is a volumetric description of the object, the set operations can be performed easily. The only prerequisite is that a neural representation of both objects implied in the set operations exists *a priori*. Given the coordinates of a 3D point, the trained model of each object will reply with a negative value if the point is inside the object or with a positive value if the point is outside. Using this property, a point will belong to the union of two objects if it is inside the first object or inside the second object. In a similar manner, a point will belong to the intersection if it is inside the first object and inside the second object. A point will belong to the difference of two objects if it is inside the first object and outside of the second (or outside the first and inside the second).

Thus, a given point will be evaluated with the two neural networks corresponding to the two objects, and an "AND" or "OR" operation will be performed between the outputs to determine if the point belongs or not to the union/difference/intersection of the two objects, as shown in Fig. 3.12.

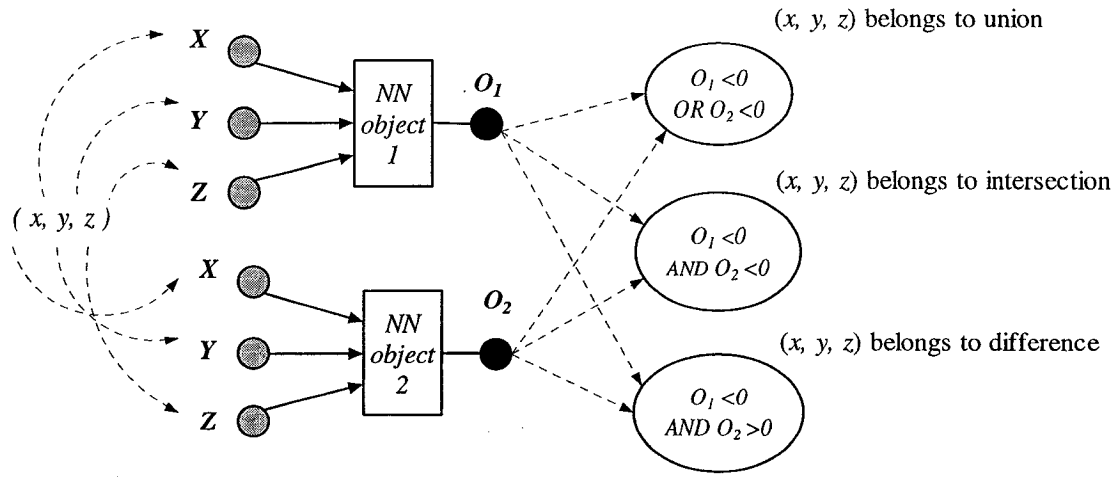


Figure 3. 12. Block diagram set operations

For the visualization of the models resulting from the set operations, the procedure is repeated for sampled points of the [−1 1 −1 1 −1 1] object space, and only the points belonging to the union/difference/intersection are plotted (section 5.1.3.2, chapter 5).

3.4.3. Object Collision

As the 3D neural representation is an implicit function, in order to detect a collision is sufficient to sample a set of points from the tested object surface and substitute them in the neural network model of the reference object. If a negative value is returned, it means that the tested points are inside of the reference object and thus a collision occurred between the two objects.

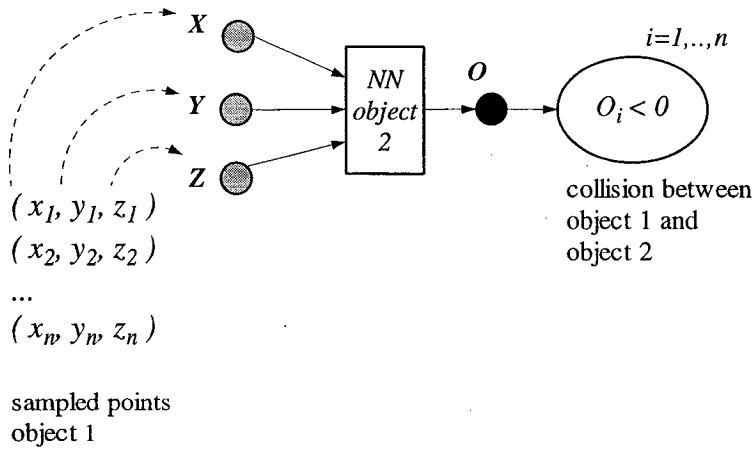


Figure 3. 13. Collision detection

The results of the tests performed for different scenes are presented in section 5.1.3.3 of chapter 5.

3.4.4. Object Classification and Recognition

One frequently encountered problem in pattern recognition is to classify a transformed/ deformed object given by a set of points, possibly affected by random noise, and to recuperate, if possible, the transformation/deformation parameters. This implies the existence of a collection (database) of reference models with which the transformed/deformed object should be identified. In order to perform this task, a neural network representation must exist for all the objects to be placed in the database. For reasons of simplicity, all these reference models are considered centred at *(0, 0, 0)*.

The model-based matching takes place without point correspondence, and is done by first aligning the transformed object with the reference model in the database using the transformation module [42]. The neural network of the transformation module is trained using the reference model points as training points (inputs) and the points of the transformed model as targets. The update of parameters is done gradually. First the scaling parameters from the affine transformation neural network are estimated, keeping constant translational and rotational ones.

Next, keeping the scaling constant, the rotation angles are computed from the weight matrix. Given the weight matrix W with m_1, m_2, m_3, n_1, n_2, n_3, q_1, q_2, q_3 given by (3.7):

$$W = \begin{bmatrix} m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \\ q_1 & q_2 & q_3 \end{bmatrix} \tag{3.44}$$

the rotation angles can be computed using:

$$\theta = \arctan\left(\frac{n_1}{m_1}\right), \quad \varphi = \arctan\left(V - \frac{q_1}{m_1 \cos\theta + n_1 \sin\theta}\right), \quad \psi = \arctan\left(\frac{q_2}{q_3}\right) \tag{3.45}$$

The translation parameters are directly recuperated from the biases' values a, b, and c, where $x_T = a$, $y_T = b$, $z_T = c$.

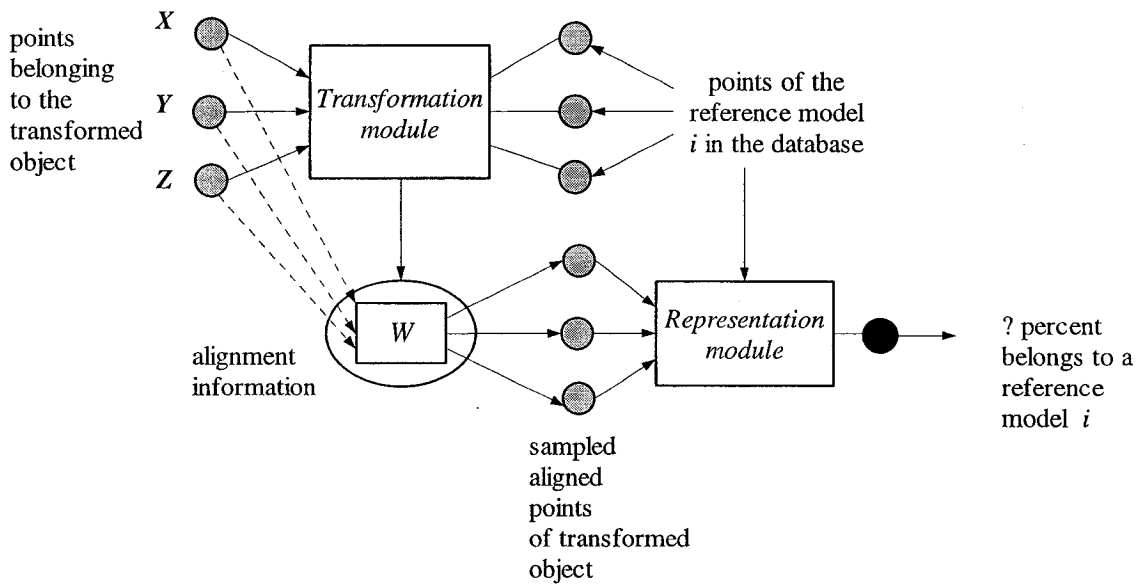


Figure 3. 14. Block diagram of classification/recognition

In this way, the transformation information is stored in the weights of the network and the two models are aligned. After the alignment procedure, a set of points is sampled from the transformed object surface and all these points are tested as if they belong or not to the reference model. A large percentage of points belonging to the reference object implies the recognition of an object in the database.

The bending, tapering or twisting parameters can be estimated in the same way. However, the recognition can become complex in this case. It is almost impossible to differentiate between different objects having somewhat similar shapes, for example between a screw-like

shape and a twisted parallelepiped. Another problem appears in case of complex scenes, with many objects. A segmentation of the scene is required prior to the recognition process. The k-means clustering algorithm is used to divide the scene in the component objects. However, errors in segmentation can detract from the recognition capabilities of the proposed system.

Test procedures have been run to check the behaviour of the system in all these cases, with and without random noise added (section 5.1.3.4, chapter 5).

3.4.5. Object Motion Estimation

Given a known 3D object already represented by a MLFF neural network, it is possible to estimate the motion based on a sequence of sets of 3D range data taken while the object is moving.

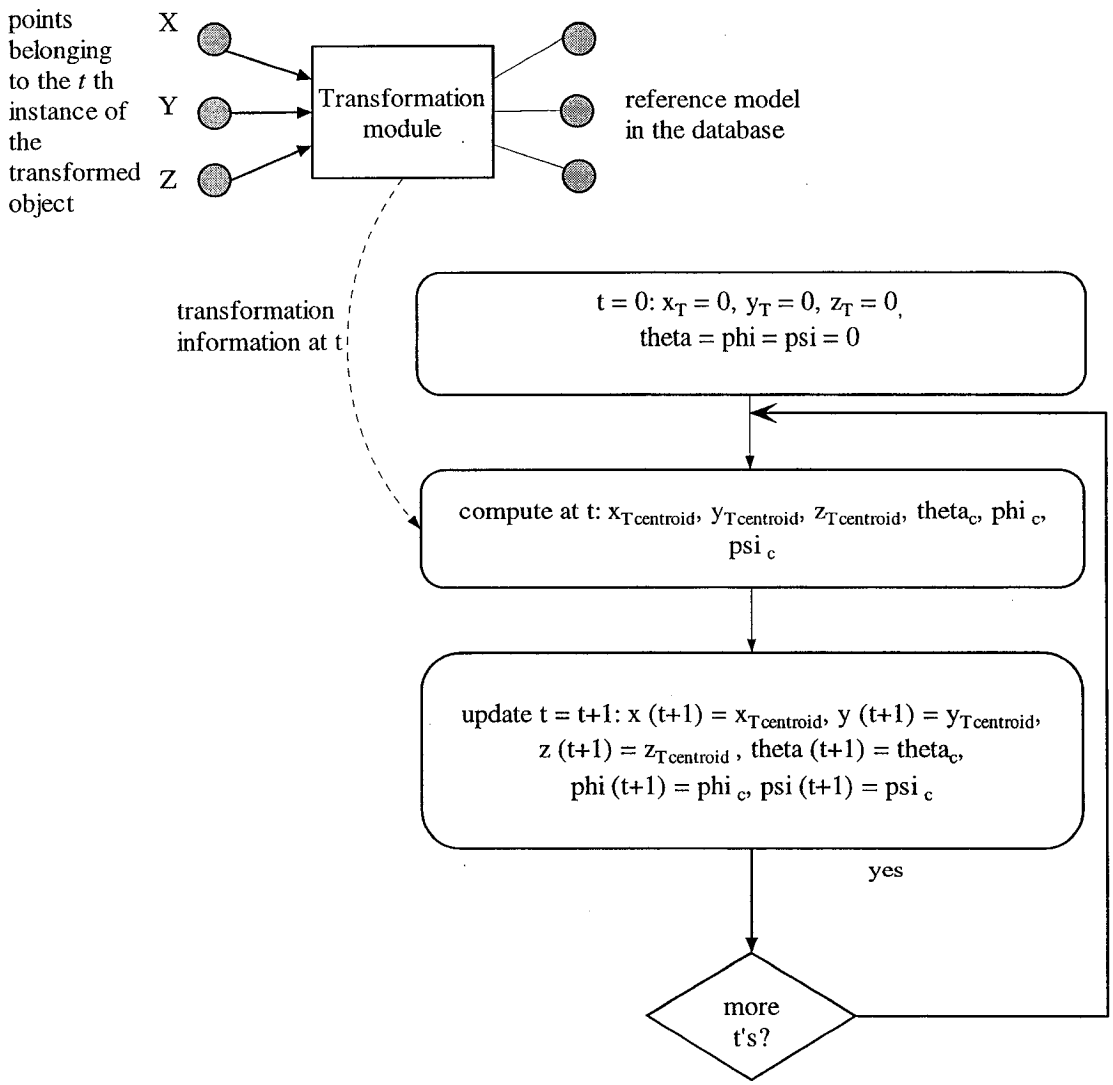


Figure 3. 15. Block diagram of motion estimation

The motion of a 3D object is described in terms of rotation, translation and scaling, as given in the TRPY generalized motion matrix. The reference object is assumed to be in the origin, and an initial guess is made for all the implied parameters. The translation and rotation parameters are initially set to 0, and the scaling factor to 1 (the size is not supposed to change). The motion parameters are computed using the transformation module as shown in section 3.4.4.

Based on the assumptions that the movement of the object and the measurements are continuous, the next translation parameter is always initialized with the position of the object's centroid in the previous step. The rotation angles are set also to their previous value (as it is unlikely that abrupt changes occurred since the last movement step). The scaling factor is set to 1 in each step. The procedure is repeated until there are no more movements in the scene. It is considered that no more movements have been made when for four consecutive steps there has been no change in any of the parameters.

Tests have been conducted (section 5.1.3.5, chapter 5) to check the ability of the system in two scenarios: first only with the translation parameters, second with both the translation, rotation and scaling parameters, in the presence of different levels of random noise.

Chapter 4

Self-Organizing Neural Network Architectures for 3D Objects Representation

This chapter describes two self-organized architectures for 3D object representation: the self-organizing map and the neural gas network. It starts by providing a brief introduction to the main ideas behind self-organization and links them with vector quantization methods. It then takes a look both to the self-organizing map and neural gas network, taking into consideration aspects such as: purpose, topology, training algorithms, data sets and testing procedures. It finally shows some possible applications of the described models.

4.1. Introduction

As opposed to the solution presented in the previous chapter, which was based on supervised learning, both architectures proposed here are unsupervised ones. In unsupervised learning the network adjusts itself by using the input patterns only. There is no target set specified, and hence the network cannot determine errors on which to base external feedback for learning. An unsupervised network can, however, group similar sets of input patterns into clusters predicted upon a predetermined set of criteria relating the components of the data [86].

From here stems the close relationship existing between vector quantization methods and unsupervised learning neural networks. Both vector quantizers and neural networks are adaptive techniques, which can be usually represented in the form of a graph of computational units and realize some kind of distribution approximation [95]. The competitive learning strategy that

characterizes unsupervised learning neural networks can be derived from vector quantization techniques. Vector quantization techniques encode a data set using a finite set of reference (codebook) vectors that constitute cluster centres. Data patterns are classified into clusters according to some similarity measure, which is usually the distance between the pattern and the nearest cluster centroids. The "winning" data vector is determined to be the closest in distance from the input vector. In the same way, a competitive learning network is a neural network in which each group (cluster) of neurons competes for the right to become active [86]. The most extreme competitive learning strategy is the "winner-takes all" where the activation of the neuron with the largest net input is the one to have its weights updated, as described in the vector quantization. In this case, however, neurons which have weight vectors that are far from any input vector may never win and hence, never learn. These are like "dead neurons". Thus the new idea of soft competition appeared.

Soft competition involves updating all the nodes in the network to some degree as opposed to just the winning node. This movement of all nodes, although computationally expensive, helps to improve the performance in two ways. It helps to prevent dead units on one side, and on the other side, the change in the value of the error function is not just in the direction of the steepest gradient but may cause a jump to a new value somewhere else in the error space, thus improving the learning procedure [95].

The self-organizing map (SOM) is one of the most important unsupervised network architectures that uses soft competition and in the same time belongs to the vector quantization methods. As in adaptive vector quantization methods, the input space is clustered by assigning each neuron to a defined region in the input space, as depicted in Fig. 4.1a. The number of inputs to each external neuron being equal to the dimension of the input space, the weight vectors can be interpreted as locations in this space. Each region is a set of locations that are closer (according to a distance measure) to the corresponding neuron than any other one. After the learning procedure, two vectors belonging to the same cluster will be projected on two close neurons in the output space. Usually the defined topology of a SOM is a two-dimensional grid.

The low-dimensional mapping topology of SOM is removed in the neural gas network [96]. The neural gas network consists in nodes (cluster centres), which independently move over the data space while learning (Fig 4.1b). The name emphasizes the similarity to the movement of gas molecules in a closed container [97]. Neural gas is a vector quantizer that converges quickly, reaches lower distortion error after convergence than other methods and obeys a gradient descent on an energy function, which is not the case of Kohonen map. It presents the identical vector

quantization property of Kohonen's algorithm but does not hold the topology preservation property.

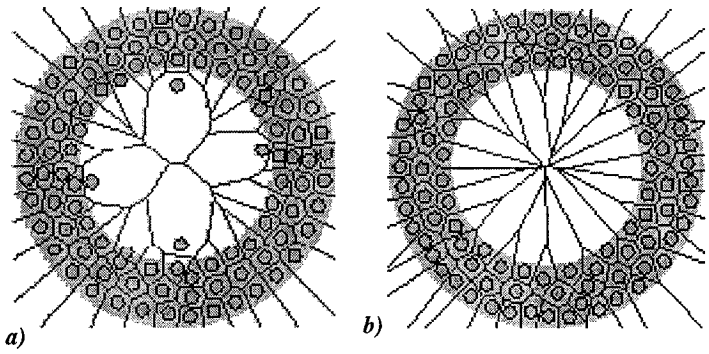


Figure 4. 1. Example of clustering with SOM and Neural Gas (reproduced from [99])

Both these two architectures will be discussed in the context of 3D object modeling.

4.2. Self-Organizing Map

4.2.1. Purpose

The main purpose of the self-organizing map is to map a continuous high-dimensional space into a discrete space of lower dimension (usually 2) [95]. In the context of this thesis it will be employed to obtain a point-based compressed model for the 3D pointcloud that represents an object. The weight vector will consist of the 3D coordinates of the object's points. During the learning procedure, the model will contract asymptotically towards the points in the input space, respecting their density and thus taking the shape of the object encoded in the pointcloud.

4.2.2. Topology

As presented in chapter 2, a SOM contains one layer of neurons and two layers of connections. Each neuron has external connections to the inputs and is also laterally connected to its neighbours up to a certain distance. Computations are feedforward in the first layer of connections and bi-directional in the connection layer. Each neuron in the first layer computes the weighted sum of the received inputs, while the purpose of the second layer of connections is to reinforce the activation values of 'strong' neurons and decrease the activation of 'weak' ones [95].

4.2.2.1. Activation Function

Self-organizing maps exploit the idea of soft competition. Given a set of p inputs, x_1, ..., x_p, and a set of N outputs y_1,..., y_N, the activation of the neuron n_j (marked with black in Fig. 4.2) is given by:

$$y_j = \sigma\left[\sum_{i=1}^{p} w_{ji} x_i + \sum_{k=-K}^{K} c_{jk} y_{j+k}\right] \tag{4.1}$$

where w_{ji} denotes the weights corresponding to input i, c_{jk} the weights corresponding to the lateral connections, K is the radius of the lateral interconnection and σ is a nonlinear function.

The solution to the nonlinear eq. (4.1) can be obtained using relaxation techniques. However, a simple interpretation is that the role of the connection layer is to reinforce the activity in the areas where strong activities of neurons already exist (using the first term of the equation) and to decrease the activations of neurons in the other areas. Due to the lateral feedback connections, the initially random activity of the map tends to form over time 'bubbles of activity' clustered around certain nodes. This means that the values of y_j tend to concentrate inside a spatially bounded cluster. The centre of this 'activity bubble' occurs at neuron n_j which is the best match between the input vector **x** and the weight w_j.

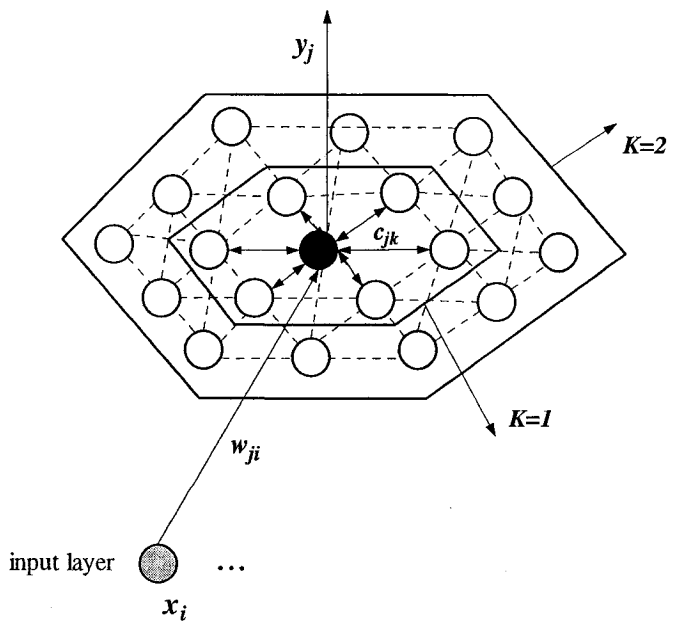


Figure 4. 2. Kohonen self-organizing map

To find the best match, the inner products $w_j\mathbf{x}$ (where $\mathbf{x}$ is the vector of inputs) are compared and the largest one is selected. But the term $w_j\mathbf{x}$ is identical to the first summing term in eq. (4.1). And thus, selecting the largest inner product, determines the location where the 'activity bubble' is to be formed.

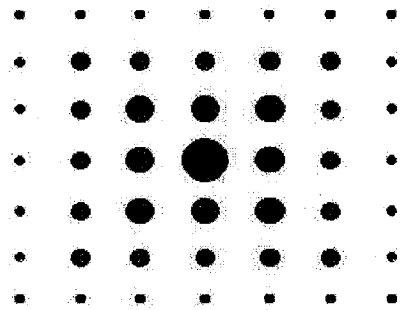


Figure 4. 3. Activity bubble

In practice, to avoid the necessity of normalization with the inner product, the Euclidean distance is used as the best-matching criterion. The index to identify the winning neuron that best matches the input vector $\mathbf{x}$ is determined using:

$$i(x) = \arg\min_j \|x - w_j\|, \quad j = 1,\ldots,N \tag{4.2}$$

where ||.|| denotes the Euclidean norm of the argument vector. It means that the neuron n_j is the winning neuron if its weight vector w_j makes the smaller angle with the input $\mathbf{x}$ on the unit circle of normalized vector than any weight vector corresponding to other neurons [86].

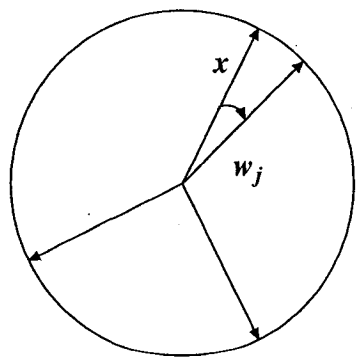


Figure 4. 4. Normalized vectors on the unit circle. The winning neuron's weight w_j makes the smaller angle with x (redrawn from [86])

This mechanism implements the competition inherent in the nature of the SOM. The selection of the winning neuron will be used in the learning process of the network.

4.2.2.2. Neighbourhood Function

A SOM is formed of neurons located on a regular grid, usually 1 or 2 dimensional as shown in Fig. 4.5.

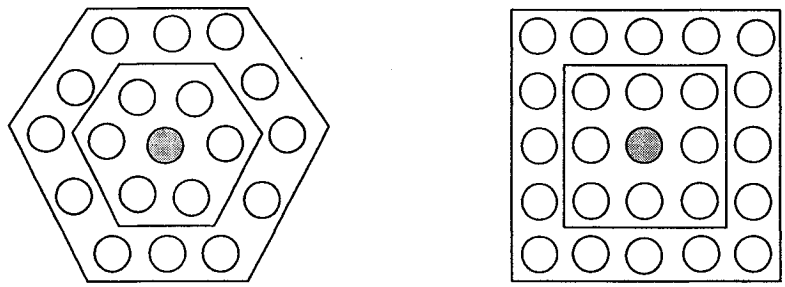


Figure 4. 5. 1 and 2 – Neighbourhoods on a hexagonal and a rectangular grid

The neurons are connected to adjacent neurons by a neighbouring relationship that determines the structure of the map. In the 2D case, the neurons are arranged either on a rectangular or a hexagonal lattice. Immediate neighbours correspond to the 1-neighbourhood. The neighbourhood function determines how strongly the neurons are connected, and thus actually controls the way neurons cooperate with each other. Usually the hexagonal grid is recommended because all the 6 neighbours are situated at the same distance, as opposed to the ones in the rectangular lattice [98].

Following the biological motivation, the lateral feedback is usually described by the Mexican hat neighbouring function (Fig. 4.6). Connections for close neurons are positive (excitatory, marked with '+'), the ones for more distant neurons are negative (inhibitory, marked with '-') and tend to 0 for far neurons.

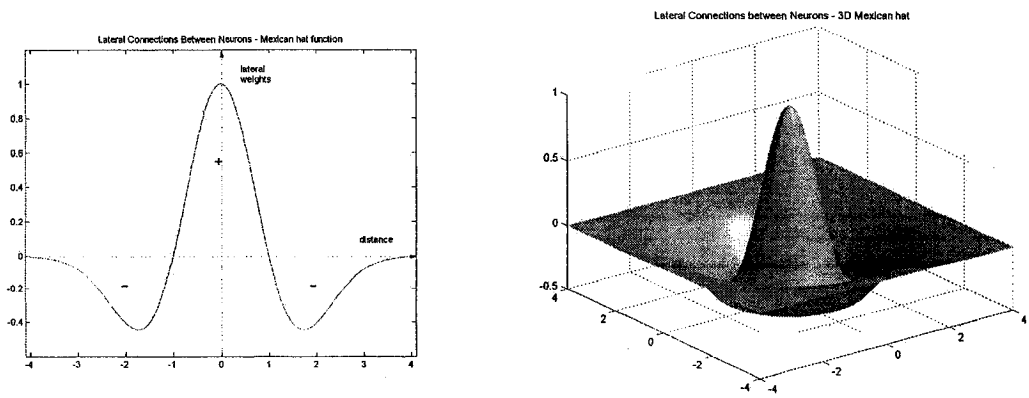


Figure 4. 6. Lateral connections between neurons (the Mexican Hat function)

The Mexican hat function is given by the equation:

$$H(x) = (1 - x^2)\exp\left(-\frac{x^2}{2}\right) \tag{4.3}$$

It is usually approximated at discretized locations [95] as in Fig. 4.7.

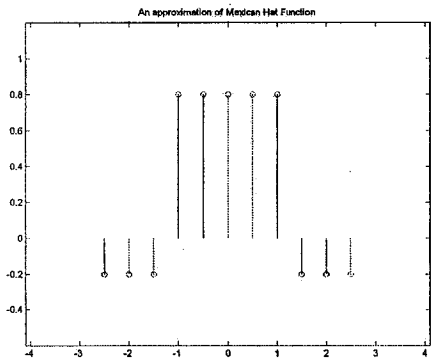


Figure 4. 7. Lateral connections between neurons (the Mexican Hat function approximation)

However, in order to simplify even more the computation of the Mexican hat, a simple Gaussian function is usually used to model the neighbourhood function. In this case there is no inhibitory effect, just an excitatory one.

The Gaussian for a 2D grid is given by:

$$h_{ij} = \exp\left(-\frac{\|r_j - r_i\|^2}{2\sigma^2}\right) \tag{4.4}$$

where $\|r_j - r_i\|^2$ denotes the lateral Euclidean distance of neuron j from the winning neuron i and σ is the neighbourhood radius. This function is more a function of distance between neurons on the grid than a function of the neighbourhood set depicted in Fig. 4.5.

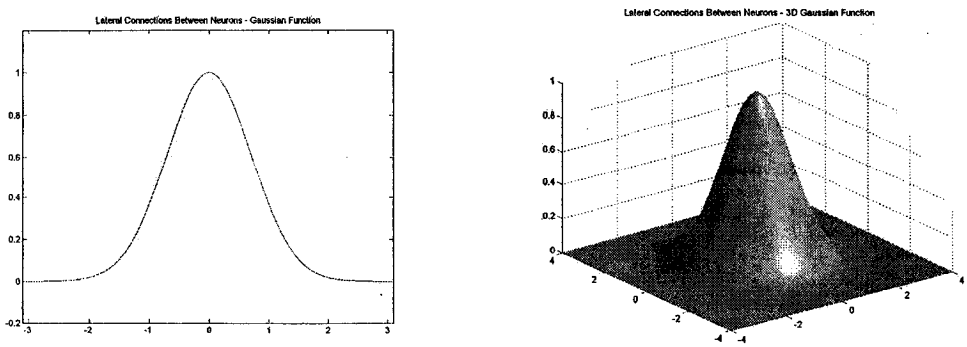


Figure 4. 8. Lateral connections between neurons (the Gaussian function)

4.2.3. Training Algorithm

The training algorithm for a self-organizing map is performed in 3 steps: weight initialization, sampling and similarity matching and updating.

4.2.3.1 Weight Initialization

Before training, some values have to be chosen to initialize the weights. The self-organizing map is in general robust with respect to initialization [98]. However, a better selection of weights can lead to faster convergence. Typically there are three ways in which the initialization can be done: randomly (with the single requirement that the magnitude of the weights is small), using random samples from the input set or using linear initialization. In the last case, the SOM is initialized linearly along its greatest eigenvectors of the input set. Of the three methods, the last one is considered to be the fastest since the SOM is already approximately organized in the beginning and a narrower neighbourhood function and a smaller learning rate can be used [98].

4.2.3.2 Sampling and Similarity Matching

In each training step, one sample vector **x** is chosen and the best matching neuron $i(x)$ at time n is computed using the minimum Euclidean distance criterion:

$$i(x) = \arg\min_j \left\| x(n) - w_j \right\|, \quad j = 1,\ldots,N$$

4.2.3.3 Updating

After finding the winning neuron, the weight vectors of the winning neuron and those of its topological neighbours are updated (moved closer to the input vector in the input space, as in Fig. 4.9).

The update rule for the neurons is:

$$w_j(n+1) = \begin{cases} w_j(n) + \eta(n) h_{ij}(n)[x(n) - w_j(n)], & when\ j \in \Lambda_{i(x)}(n) \\ w_j(n), & , \quad otherwise \end{cases} \tag{4.5}$$

where n denotes the discrete time, $\eta(n)$ is the learning-rate parameter and $h_{ij}(n)$ is the Gaussian function representing the topological neighbourhood $\Lambda_{i(x)}(n)$ centred around the winning neuron $i(x)$, as described in eq. (4.2).

The algorithm is repeated with the sampling of the next input vector until no noticeable changes occur in the map. The learning rule is local, as the operation at each neuron only requires the knowledge of the weight associated with that neuron and the input vector. However, a non-local function is needed to choose the best-matching neuron [99]. For the batch algorithm, the only difference is that the whole data set is run through at once and the map is updated with the effect of all samples [98].

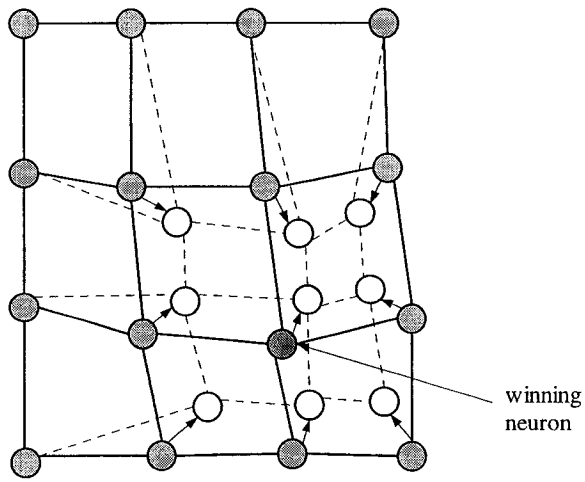


Figure 4. 9. Updating the winning neuron and the neighbours (redrawn from [98])

Derivation of the Learning Rules

For the network to be self-organizing, the weight w_j of a neuron n_j is required to change in relation with the input **x** [92]. The learning occurs according to Hebbian postulate of learning by adapting:

$$\frac{dw_j}{dt} = \eta y_j x \tag{4.6}$$

where η is the learning-rate.

However, since the inputs are excitatory (positive), the use of this equation would allow for adaptation in the positive direction only. To overcome this problem, a second "forgetting" term is added $-g(y_j)w_j$, where $g(y_j)$ is a positive scalar function. But as a consequence of the 'activity bubble' phenomenon, the response y_j of the neuron j can be considered 1 if the neuron j is active (inside the neighbourhood $\Lambda_{i(x)}$) or 0 if is inactive (outside the neighbourhood). In a similar way, the function $g(y_j)$ can return a positive constant value if the neuron is active and 0 otherwise [92]. Without loss of generality, this positive constant can be made equal to η, and thus:

$$\frac{dw_j}{dt} = \begin{cases} \eta(x - w_j), & \text{when neuron } j \text{ is in the neighbourhood } \Lambda_{i(x)} \\ 0, & \text{otherwise} \end{cases} \quad (4.7)$$

Eq. (4.7) becomes in discrete time:

$$w_j(n+1) = \begin{cases} w_j(n) + \eta(n)[x(n) - w_j(n)], & \text{when } j \in \Lambda_{i(x)}(n) \\ w_j(n), & \text{otherwise} \end{cases} \quad (4.8)$$

and given the Gaussian neighbourhood function is used, it can be re-written as:

$$w_j(n+1) = \begin{cases} w_j(n) + \eta(n) h_{ij}(n)[x(n) - w_j(n)], & \text{when } j \in \Lambda_{i(x)}(n) \\ w_j(n), & \text{otherwise} \end{cases}$$

which gives the justification for using eq. (4.5) for updating.

Selection of Parameters

The learning rate $\eta(n)$ and the neighbourhood function $h_{ij}(n)$ are both time-dependent. In order to ensure that the algorithm converges, is important that the learning rate parameter and the neighbourhood radius σ are decreased slowly during the learning process. Although a linear function can be used to describe the decrease in time, the most popular choice is to use it in form of a power series:

$$\sigma(n) = \sigma_0 \left(\frac{\sigma_T}{\sigma_0} \right)^{n/T} \quad (4.9)$$

$$\eta(n) = \eta_0 \left(\frac{\eta_T}{\eta_0} \right)^{n/T} \quad (4.10)$$

where the constants σ_0 and η_0 are the initial values for $\sigma(n)$ and $\eta(n)$, σ_T and η_T are the final values, n is the time step and T the training length.

4.2.4. Self -Organizing Map Properties

There are three main advantages of self-organizing maps:

1. *Input Space Approximation* - The SOM is able to preserve the structure of the input space relatively well. This property is very important for data reduction, and is at the same time the goal of vector quantization.

2. *Topology Ordering* – A SOM transforms or preserves the similarity between input vectors from the input space into topological closeness of neurons in the output space. Each output neuron specializes during the training procedure to react similar to similar input vectors (belonging to a cluster in the input space) [100]. The neurons in the SOM output are topologically ordered in the sense that neighbouring neurons correspond to similar regions in the input space.
3. *Density Matching* - The SOM preserves the data densities of the input space reasonably well, that is, regions with more data points are mapped to larger regions in the output space.

However, to obtain good results with a Kohonen SOM, the topology of the input space to be represented has to match the grid used. As we previously stated, usually maps of higher dimension than 2 are not used since their visualization is problematic [101]. Moreover, the SOM algorithm does not have a cost function in the general case that yields the adaptation rule used (eq. 4.5) as its gradient [96]. To overcome these problems, a neural gas network releases the neurons from the 2-dimensional grid and obeys a gradient descent on a cost function surface.

4.3. Neural Gas Network

4.3.1 Purpose

The main purpose of the neural gas network is to cluster multi-dimensional vectors. An initial large set of vectors is reduced to a smaller set of neurons of the same dimension, with a minimal mean distortion between each of the vectors in the initial set and its best-matching neuron. The best matching neuron is defined as the vector in the final set that is the closest to the input vector [95]. In the context of this thesis, the neural gas network will be employed like the SOM, to model the 3D pointcloud that represents an object.

4.3.2 Topology

The neural gas network does not use a particular topology. It consists of a number of disconnected centres. It can be however seen as a feedforward unsupervised model having one layer of neurons and one layer of connections [95], as depicted in Fig. 4.10. Each neuron is connected to the input vector of the same dimension, but there are no connections between units in the neuron layer.

4.3.2.1. Activation Function

The neural gas network uses the same soft competition as in the case of the SOM. The index to identify the winning neuron that best matches the input vector x is determined using:

$$i(x) = \arg\min_j \|x - w_j\|, \quad j = 1,\ldots,N \tag{4.11}$$

where ||.|| denotes the Euclidean norm of the argument vector. However, the neurons to be adapted in the learning procedure are not selected according to a topological relation, but rather according to their rank.

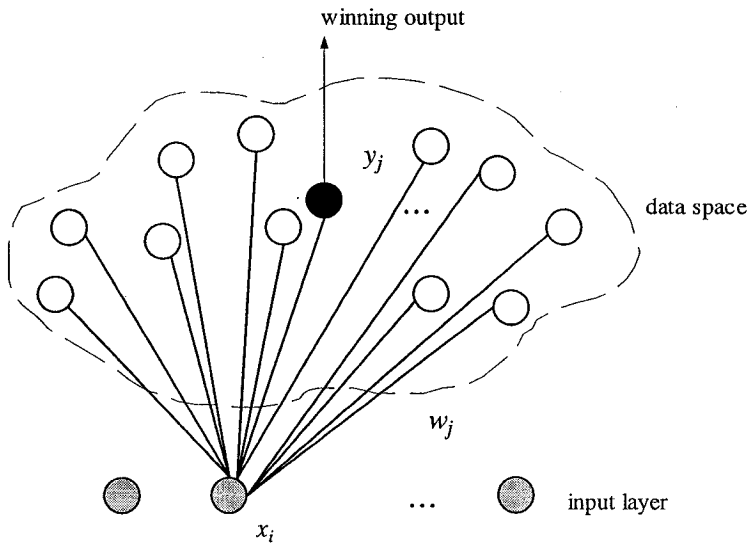


Figure 4. 10. Neural gas topology

4.3.2.2. Neighbourhood Ranking

The rank of neurons is the rank they have in an ordered list of distances between their weights and the input vector. Each time a new input vector $x(n)$ is presented to the network, a neighbourhood ranking indices list is built $(j_0, \ldots, j_{N-1})$, where w_{j0} is the weight of the closest neuron to x, w_{j1} the weight of the second-closest neuron, and w_{jk} is the reference vector such that k vectors w_i exist with:

$$\|x - w_i\| \leq \|x - w_{j_k}\| \tag{4.12}$$

4.3.3. Training Algorithm

4.3.3.1 Weight Initialization

The weight initialization is done with very small values, of the order of 10^{-5}. The neural gas is in general invariant to initialization. This is due to the fact that the distance in input space is only used for determining the rank order, and then the adaptations are done on the basis of this rank order [99].

4.3.3.2 Sampling and Neighbourhood Ranking

In each training step, one sample vector **x** is chosen and the best matching neuron $i(x)$ at time n is computed using the minimum Euclidean distance criterion (eq. (4.11)). All neurons are then ordered according to eq. (4.12). Based on this rank, a certain number of units is adapted.

4.3.3.3 Updating

The neurons' weights are updated according to:

$$w_j(n+1) = w_j(n) + \alpha(n) h_\lambda(k_j(x, w_j))[x(n) - w_j(n)] \tag{4.13}$$

where $\alpha(n) \in [0, 1]$ describes the overall extent of the modification, and h_λ is 1 for $k_j(x, w_j) = 0$ and decays to zero for higher values according to eq. (4.14):

$$h_\lambda(k_j(x, w_j)) = \exp\left(-\frac{k_j(x, w_j)}{\lambda(n)}\right) \tag{4.14}$$

As in Martinetz *et al.* [96], $k_j(x, w_j)$ is a function that represents the ranking of each weight vector w_j. If j is the closest to input **x** then $k = 0$, for the second closest $k = 1$ and so on. As in the case of SOM, the learning rule is local (the operation at neuron level only requires the knowledge of the weight associated with that neuron and the input vector). However, a non-local function is needed to rank the neurons [95].

Derivation of the Learning Rules

The name "neural gas" emphasizes the similarity to the movement of gas molecules in a closed container. Thus a neural gas network can be seen as using a temperature coefficient to affect the softness of the competition over time [95]. The network minimizes the following cost (error) function:

$$E(w,\lambda) = \frac{1}{2C(\lambda)} \sum_{j=1}^{N} \int P(x(n)) h_\lambda(k_j(x,w)) \cdot (x - w_j)^2 \tag{4.15}$$

where $P(x)$ is the distribution of input vectors $x(n)$ and

$$C(\lambda) = \sum_{k=0}^{N-1} h_\lambda(k) \tag{4.16}$$

The parameter λ can be seen as a temperature factor. At high temperatures many of the nodes in the network add to the cost of the network. As the temperature is reduced only the nodes close to the input contribute significantly to the error, and at very low temperatures only the winning node produces a cost.

The corresponding update rule, that is applied to all nodes and which reduces the cost function in eq. (4.15), is:

$$w_j(n+1) = w_j(n) + \alpha(n) h_\lambda(k_j(x,w_j))[x(n) - w_j(n)]$$

as in the updating equation (4.13).

Selection of Parameters

The learning rate $\alpha(n)$ and the function $\lambda(n)$ are both time-dependent. In order to ensure that the algorithm converges, it is important that these parameters are decreased slowly during the learning process. Usually the following time dependencies are used:

$$\alpha(n) = \alpha_0 \left(\frac{\alpha_T}{\alpha_0} \right)^{n/T} \tag{4.17}$$

$$\lambda(n) = \lambda_0 \left(\frac{\lambda_T}{\lambda_0} \right)^{n/T} \tag{4.18}$$

where the constants α_0 and λ_0 are the initial values for $\alpha(n)$ and $\lambda(n)$, α_T and λ_T are the final values, n is the time step and T the training length.

4.3.4. Neural Gas Properties

A neural gas network is claimed to have three main advantages over other methods [96]:

1. It converges quickly.
2. It is more accurate (reaches a lower distortion error after convergence).

3. It obeys a gradient descent on a cost function, given by eq. (4.15).

However, the neural gas network can be computationally expensive. This cost is mainly due to the neighbourhood ranking procedure. All nodes have to be ranked and updated for each input vector.

4.4. Data Sets

In this thesis, range data pointclouds and also sets of synthetic data obtained with a 3D modeling tool are used to train the network. Normalization is employed, as in the case of multilayer-feedforward to remove redundant information from a data set, by a linear rescaling of the input vectors such that their variance is 1. As in [83], for each input vector **x**, its mean is computed:

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4.19}$$

where n ($n = 1,\ldots,N$) denotes the dimension of the vector **x**, and the variance according to:

$$\sigma^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \bar{x}) \tag{4.20}$$

The set of rescaled variables are defined as:

$$\tilde{x} = \frac{x - \bar{x}}{\sigma} \tag{4.21}$$

This ensures that the transformed variables will have 0 mean and a variance of 1.

4.5. Testing

In order to evaluate the quality of the models, two simplistic measures of the resolution and topology preservation are used. The map resolution can be estimated as the average distance between each data vector and its winning neuron (*qe*, eq. (4.22)). The topology preservation can be measured as the topographic error (*te*, eq. (4.23)), which is the proportion of all data vectors for which the first and second best-matching neurons are not adjacent units [101].

$$qe = \frac{1}{N}\sum_{j=1}^{N}\left\|x_j - w_i(x_j)\right\| \tag{4.22}$$

$$te = \frac{1}{N}\sum_{j=1}^{N} u(x_j), \quad where\ u(x) = \begin{cases} 1 & if\ 1st\ and\ 2nd\ winning\ neurons\ are\ not\ adjacent \\ 0 & otherwise \end{cases} \tag{4.23}$$

The results of training and testing of both the SOM and neural gas network are the subject of section 5.2.1, 5.2.2 and 5.2.3 of chapter 5.

4.6. Applications

The SOM and neural gas are used as representation modules. As described in the previous chapter, the representation module can be cascaded with the transformation module (exactly like the one presented in the previous chapter) to perform object morphing and object motion estimation. Since the representation given by SOM and neural gas is not volumetric, there is no easy way to detect collisions, model set operations or perform object matching.

4.6.1. Object Morphing

Object morphing can be performed as described in chapter 3.4.2. The two neural networks representing the reference and the target model store in their weights the description of each object and a linear interpolation is performed between these weights to obtain the morphed objects. The same conditions must be fulfilled in order to perform the morphing operations: the objects must be aligned and modelled using the same number of neurons. The results are presented in section 5.2.3.1, chapter 5.

4.6.2. Object Motion Estimation

Making the assumptions that measurements are continuous and that the exemplar object is initially situated in the origin, the motion is estimated based on a sequence of sets of 3D range data taken during the object's movement. The motion is also considered to be continuous from one movement to the other. The transformation module is provided with the point of the reference model as inputs and with those of the moving object at certain moments in time. Motion parameters are recuperated at each moment from the weights of the transformation module.

4.6.3. Object Segmentation

As both the SOM and the neural gas network are in fact also vector quantization methods, they are able to preserve the structure of the input space and at the same time reduce the data set. This property makes them useful as a preprocessing technique on data representing complex scenes, prior to clustering. Tests are run (section 5.2.4.3, chapter 5) with just k-means clustering and then with the neural architectures prior to the use of k-means clustering to elicit the improvements given by them in the clustering procedure.

Chapter 5

Results and Evaluation

The purpose of this chapter is to present the experimental results for the three neural network architectures presented in the last two chapters. It discusses implementation issues, parameters' adjustments and the modeling results in terms of purpose, computational cost, complexity, conformance and convenience and ease of manipulation. It shows how the models can be used in the virtualized reality context. The chapter ends up with a critical comparison between the three presented neural architectures.

5.1. Multilayer Feedforward Neural Network Models

5.1.1. Evaluation of Representation Module

Matlab version 6.12, accompanied by the Neural Network Toolbox and SOM Toolbox are used to perform all the simulations in this thesis, on a Pentium III PC, 933MHz with256MB RAM. As stated in chapter 3, the multilayer feedforward architecture consists of two modules, each presenting two working modes.

5.1.1.1 Generation Mode

The representation module can be used in generation mode to obtain models for simple objects whose equations are known, as in Fig. 5.1. The values of weights and biases are set such that the network implements the desired equation of the object. For example, to build a cone centred at (0, 0, 0) and whose equation is:

$$\frac{x^2}{4}+\frac{y^2}{4}-\frac{z^2}{4}=0$$

the corresponding weights are set

$$w_{11}=1/4,\quad w_{21}=1/4,\quad w_{31}=-1/4$$

and the value of bias $b=0$.

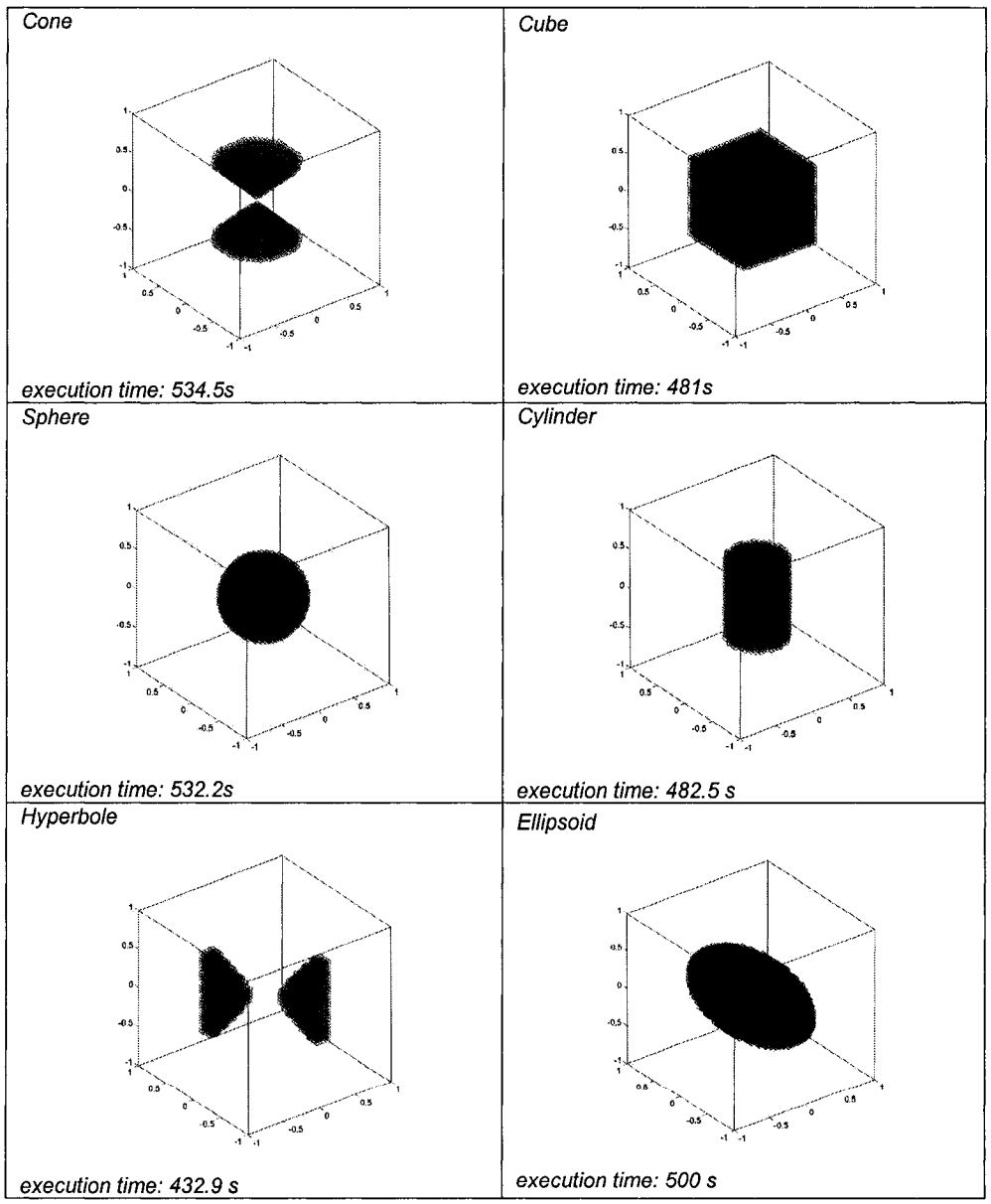


Figure 5.1. Models using representation module in generation model

Such a network will reply with a positive value if the point given as input is outside the cone and with a negative value if the point is contained in the cone. The entire object is therefore

built by evaluating the network for points in the [–1 1 –1 1 –1 1] object space in an orderly fashion, at a given step of 25. Thus the corresponding models are "normalized" in the [–1 1] interval. Only those points that return at output a value less than 0 are plotted.

The procedure is the same for all models presented in Fig.5.1. It takes approximately 8 minutes to build a model. Each such model can be further used as a primitive to build other shapes using set operations and/or morphing.

5.1.1.2. Training Mode

In training mode the representation module will be employed to learn a volumetric representation of each object given in form of pointclouds.

Data Sets

Synthetic data obtained using a 3D modeling tool and real range data from the NRC database are used to test the MLFF neural network.

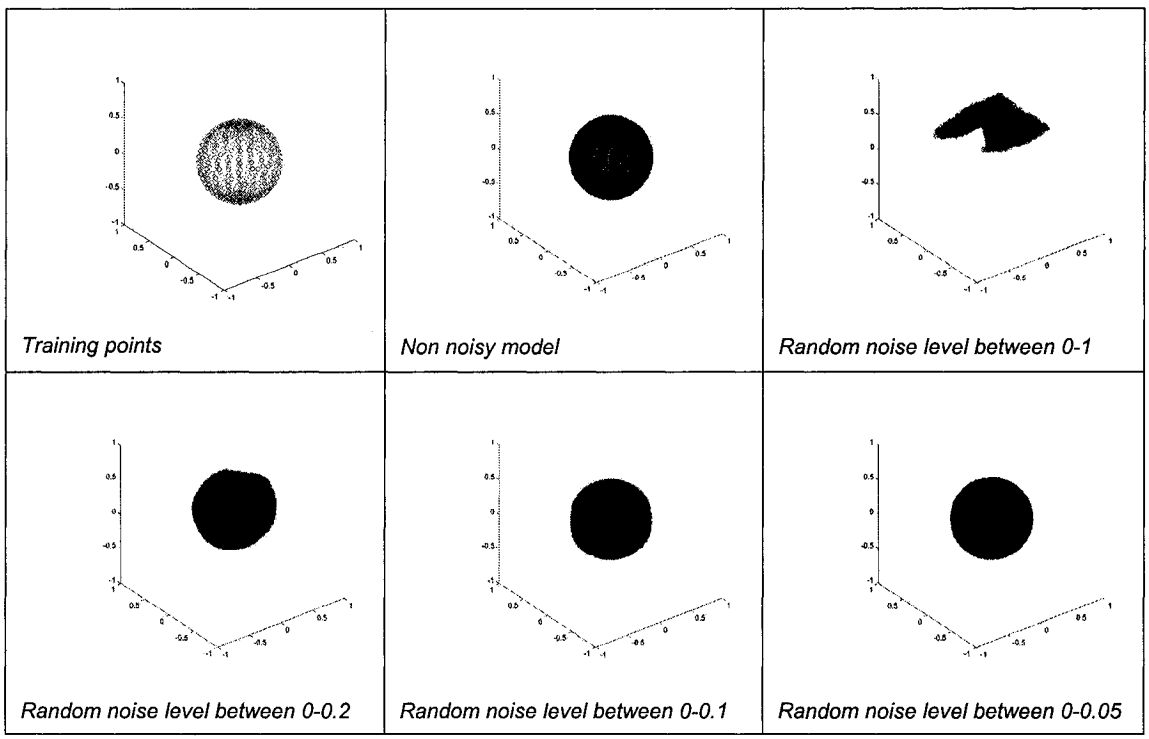


Figure 5.2. Influence of noisy data on the learning

All the points that represent a given object (or scene) are normalized in [–1 1] interval.

It is important that the training data is non-noisy or with a low level of noise. In order to evaluate noise influence on the training procedure, the MLFF neural network is trained using points belonging on a sphere corrupted with different levels of noise. The network is then tested

for points in the [–1 1 –1 1 –1 1] object space and only those points that return an output value less than 0 are plotted. The results are displayed in Fig. 5.2. For high noise levels, the network is not able to learn the shape of the object, no matter how many training epochs are used or how the learning parameters are adjusted. For a medium level of noise, the network learns only partially the shape and for a low level of noise (under 0.05), the modeling capability of the MLFF network is not affected by noise. Therefore, it is preferable to use non-noisy data or data with low-level of noise to train the network architecture.

Extrasurfaces

Using only the points of the given object for the training of the representation module is not appropriate, because the object surface is a set of zeros in the output of the neural model. In order to avoid the trivial solution and improve the training process, extrasurface points are added in the interior and exterior of the object at an equal (given) distance from the object surface along the surface normals, as shown in Fig.5.3.

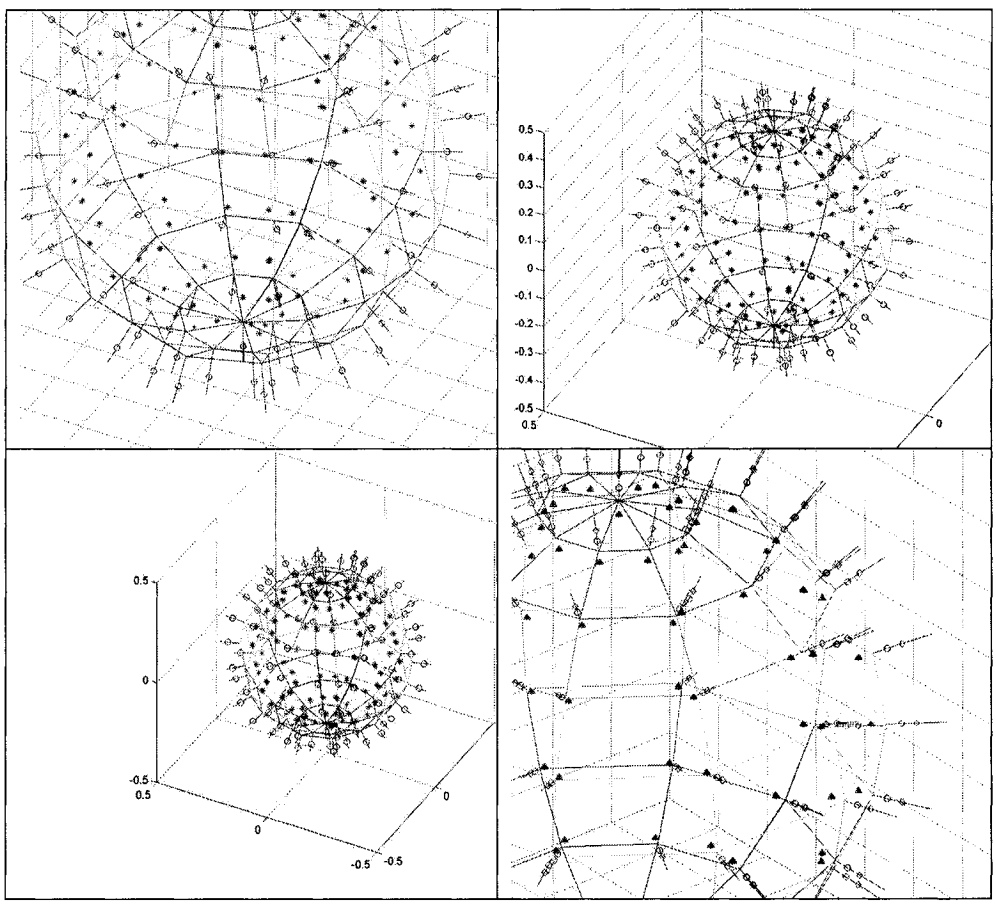


Figure 5.3. Choosing extrasurface points

Fig. 5.4 shows the layered learning of MLFF neural network corresponding to each extrasurface (3 in the shown example).

In this case the user has to decide on which inputs to use, the number of neurons in the hidden layer and the values for the training parameters. Intuitively, an object with only flat surfaces will use *X, Y, Z* and one with curved surface will use X^2, Y^2, Z^2 and the combinations of *X, Y* and *Z*. When no *a priori* knowledge is available on the shape of the object, at least *X, Y, Z* and X^2, Y^2, Z^2 should be used. During tests, we observe that objects of whatever complexity can be obtained just using these 6 inputs. The purpose of the combinations *XY*, *XZ*, *YZ* is thus restricted to modeling of objects whose equations are known, in generation mode.

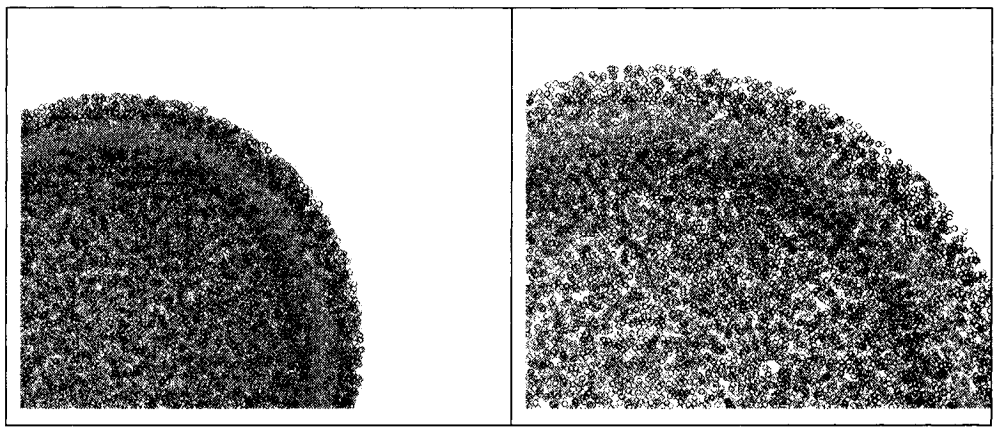


Figure 5.4. Layered Extrasurface Learning

Regarding the number of neurons in the hidden layer, tests were run with up to 20 neurons in the first hidden layer. As a rule of thumb, a good choice for the number of neurons in the second hidden layer is half the number of the neurons in the first hidden layer. Therefore, up to 10 neurons in the second hidden layer were used during tests.

The number of extrasurfaces used to model an object depends much on the object complexity and the number of points that describe the object. While the network is able to accurately learn a sphere given by 520 points with just 1 extrasurface (Fig.5.7a), it needs 5 to learn the volumetric description of the hand (Fig. 5.7d) given by 19080 points. Up to 10 extrasurfaces were used during simulations. However, we observed that 6 extrasurfaces were enough to model complex objects.

The distance at which the extrasurfaces are built is another critical factor for the learning. A too small value for this distance tends to create rounded edges, while a too large value makes it impossible for the network to learn the given object (Fig.5.5). In the tested case the value of this distance varied between 0.01 and 0.5. Special care must be taken for the case of objects with

holes (Fig.5.7b) or separated parts (Fig.5.7d, e). The distance must be chosen such that the extrasurfaces do not touch the object surface and do not touch each other.

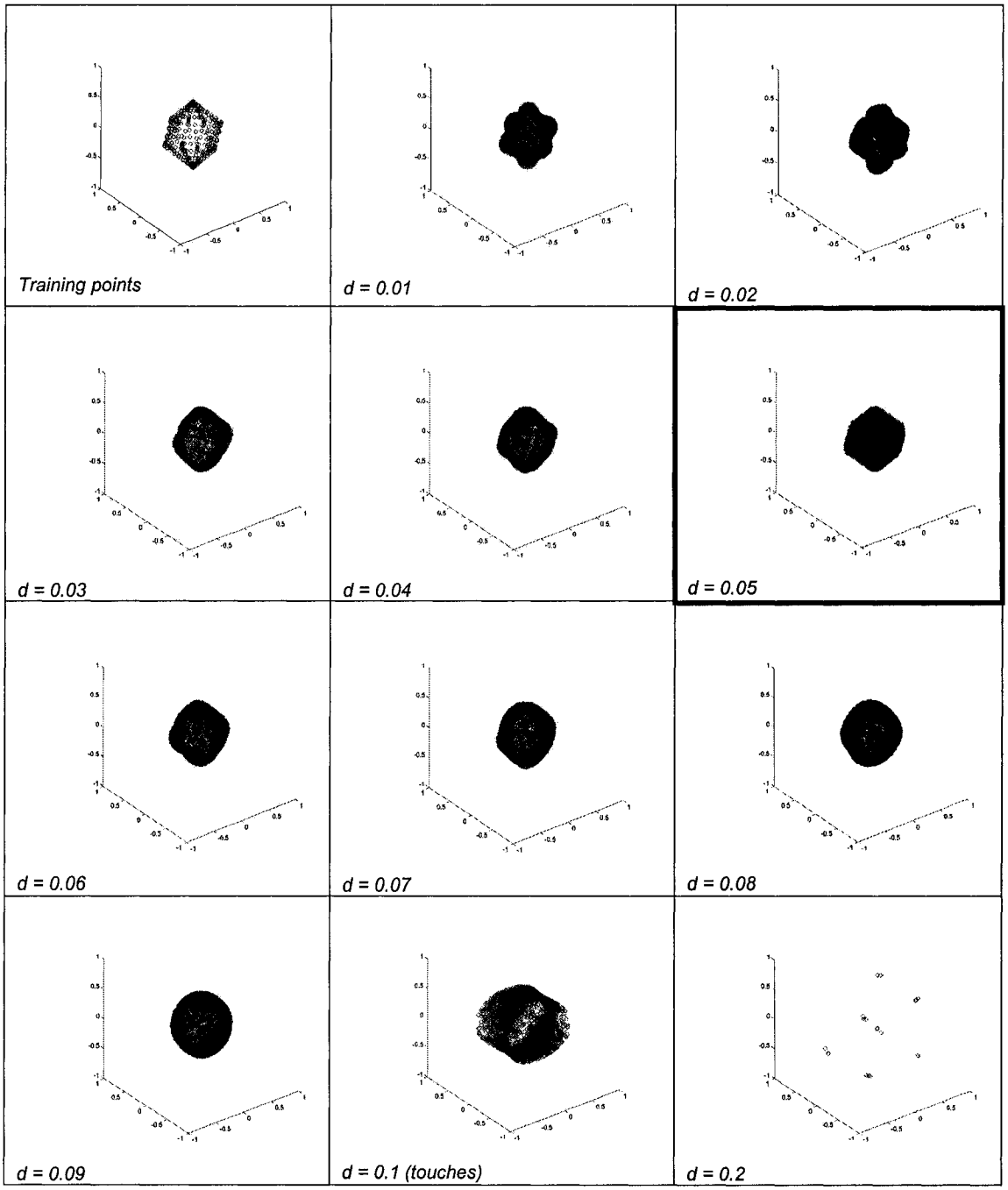
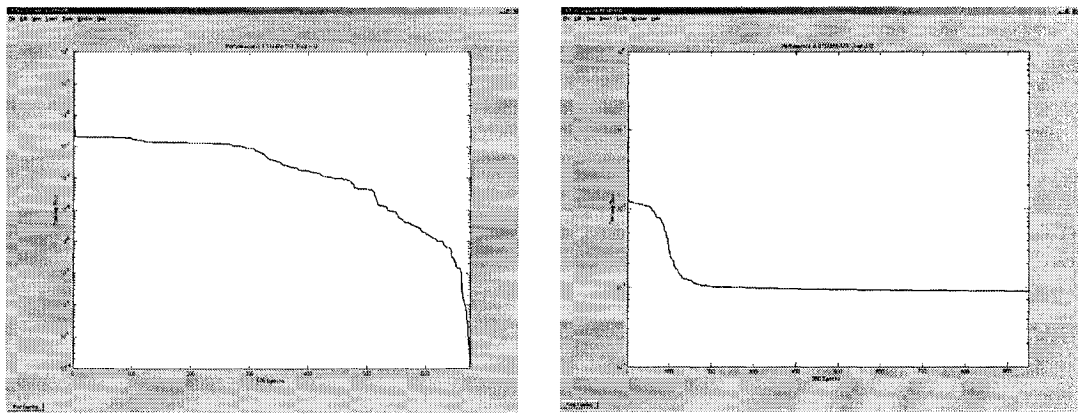


Figure 5.5. Influence of extrasurface distance (7 hidden layers, 950 epochs, 20 points, 3 extrasurfaces)

What is also worth mentioning here is the fact that even if trained with a surface, the representation obtained by the MLFF neural network is a volumetric one.

Learning and Parameter Adjustments

The module is trained using the batch version of scaled conjugate gradient backpropagation, for up to 2000 training epochs, and a mean square error goal of 0. Weights and biases are initialized using Nguyen-Widrow method, as discussed in section 3.4.1.1. The selected value of the parameter σ of the sigmoid activation function is set to 5, and the biases for the first and second hidden layers are –4 and –0.5 respectively. The parameter σ to approximate the second order information (eq. 3.36, chapter 3) is set to a very low value $5 \cdot 10^{-5}$, and the same for the λ term that is added to make the Hessian positive ($5 \cdot 10^{-7}$).



a) b)

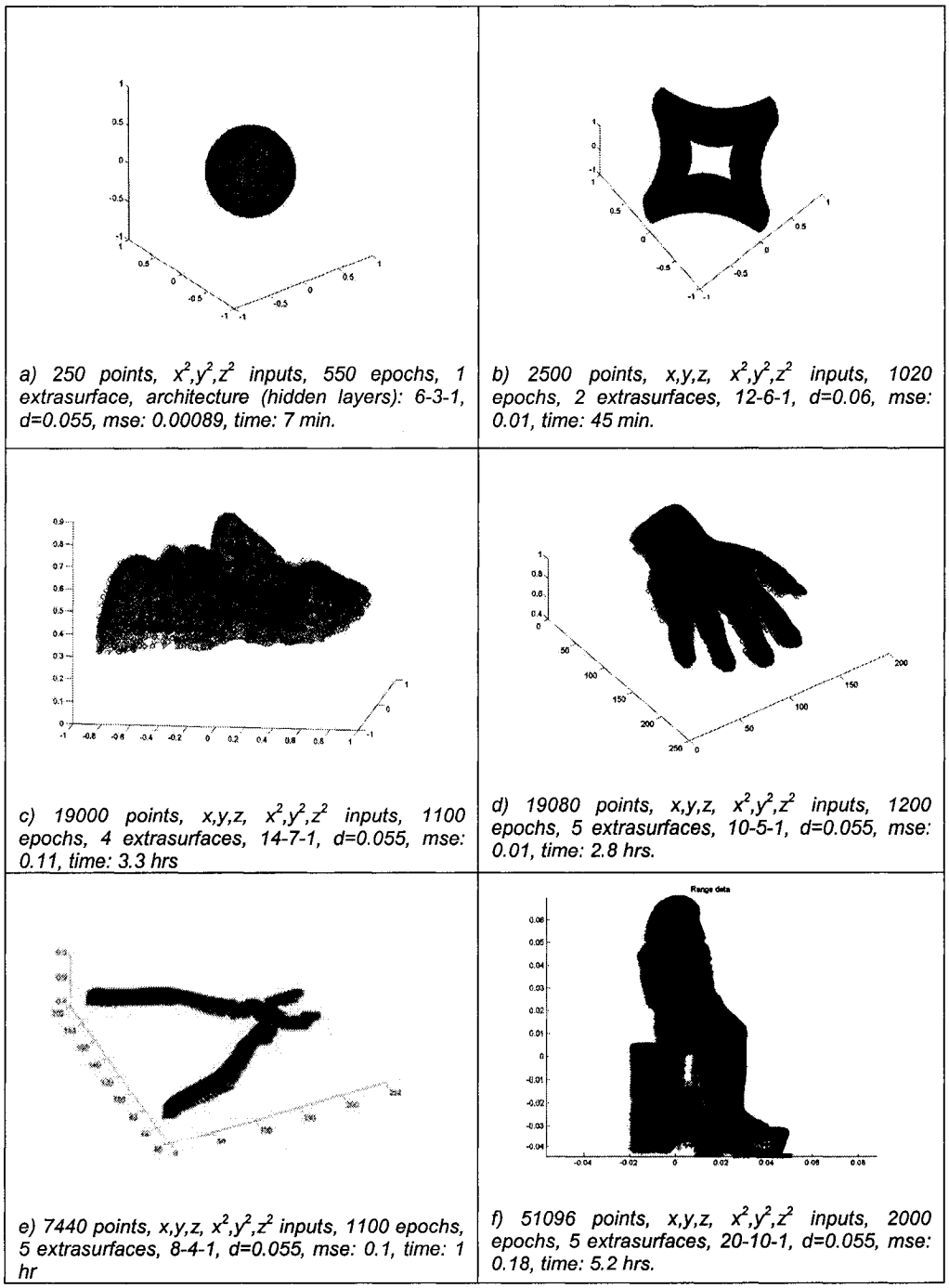
Figure 5.6. (a) Learning diagram for MLFF neural network trained with scaled conjugate gradient backpropagation. (b) False local minima situation

As all backpropagation algorithms, scaled conjugate gradient falls sometimes into false minima (Fig.5.6b), and the training has to be stopped and restarted in this case.

Fig.5.7 shows some results of training. The models in Fig. 5.7a and 5.7b are trained using synthetic data, while those in Fig. 5.7c-f are trained using real range data. In Fig. 5.7c, 5.7d and 5.7e the range data is from a single point of view. Thus, after the data is denormalized, values below a limit are cut to avoid displaying unnecessary points. These unnecessary points appear due to the fact that the representation provided by the neural network is volumetric and there are no other surfaces to limit it. The representation in Fig. 5.7f is obtained using a pointcloud that describes the object in its entirety.

The presented results demonstrate first that the training mode of the representation module is computationally expensive. This is not a significant drawback because the training occurs only once and after that the object can be used as a primitive. The generation time (the time taken to built the model from the weights) is much shorter, as depicted in Table 5.1.

On the other hand, the storage requirements are reduced in comparison with those of the representations of the same objects using polygonal models for example. Instead of using 19080 points and as many normals, only the 119 weights and the network structure (2 digits for the hidden number layers) need to be stored for the neural representation of a hand as in Fig.5.7d.



a) 250 points, x^2,y^2,z^2 inputs, 550 epochs, 1 extrasurface, architecture (hidden layers): 6-3-1, d=0.055, mse: 0.00089, time: 7 min.

b) 2500 points, x,y,z, x^2,y^2,z^2 inputs, 1020 epochs, 2 extrasurfaces, 12-6-1, d=0.06, mse: 0.01, time: 45 min.

c) 19000 points, x,y,z, x^2,y^2,z^2 inputs, 1100 epochs, 4 extrasurfaces, 14-7-1, d=0.055, mse: 0.11, time: 3.3 hrs

d) 19080 points, x,y,z, x^2,y^2,z^2 inputs, 1200 epochs, 5 extrasurfaces, 10-5-1, d=0.055, mse: 0.01, time: 2.8 hrs.

e) 7440 points, x,y,z, x^2,y^2,z^2 inputs, 1100 epochs, 5 extrasurfaces, 8-4-1, d=0.055, mse: 0.1, time: 1 hr

f) 51096 points, x,y,z, x^2,y^2,z^2 inputs, 2000 epochs, 5 extrasurfaces, 20-10-1, d=0.055, mse: 0.18, time: 5.2 hrs.

Figure 5.7. Models built with the MLFF neural network (synthetic and range data)

The resulting models are also accurate. The mean error has low values, as depicted in the last column of Table 5.1.

Table 5.1 Modeling results using a MLFF neural network

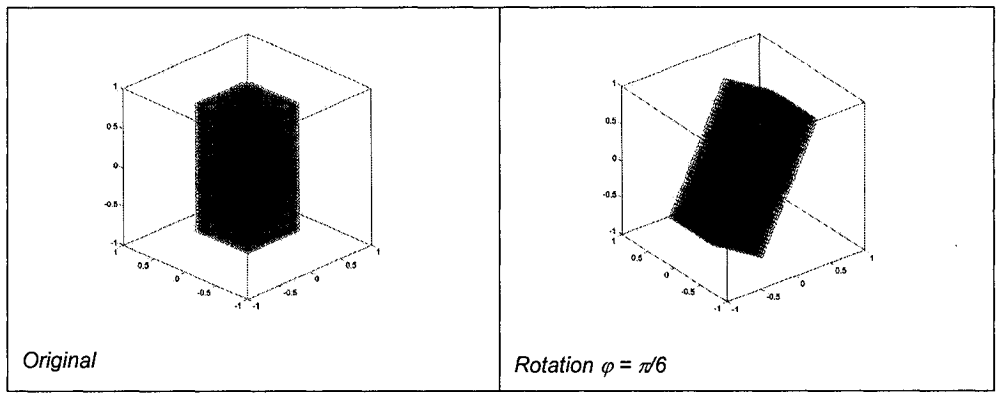
Object	*Number of Points*	*Number of Extrasurfaces*	*Number of Inputs*	*Training time*	*Generation Time*	*Number of Weights*	*Mean error (cross-validation)*
Sphere	250	1	3	7 min.	2.8 min.	44	0.14
Torus	2500	2	6	45 min.	7 min.	140	0.39
Face	19000	4	6	3.3 hrs.	8 min.	161	0.4
Hand	19080	5	6	2.8 hrs.	7.2 min.	119	0.35
Pliers	7440	5	6	1 hr.	6.5 min	98	0.24
Statue	51096	5	6	5.2 hrs	10 min.	224	0.67

5.1.2. Evaluation of Transformation Module

5.1.2.1 Generation Mode

The *transformation module* implements transformations such as rotation, translation, scaling and deformations such as bending, tapering, and twisting. It receives as inputs the 3-D coordinates *x, y, z* of a point and their combinations and returns the 3-D coordinates of the corresponding transformed point. If this procedure is repeated for each point of an object, the transformed object can be easily obtained by plotting all the transformed points.

Fig.5.8 to Fig.5.12 show some examples of affine transformation, bending, tapering and twisting results, as discussed in section 3.2.2.3 using both synthetic and real range data. Our experiments demonstrated the capability of the MLFF neural network to perform these tasks. It takes approximately 20-30 seconds to build a transformed model. These models can then be used as modeling primitives.



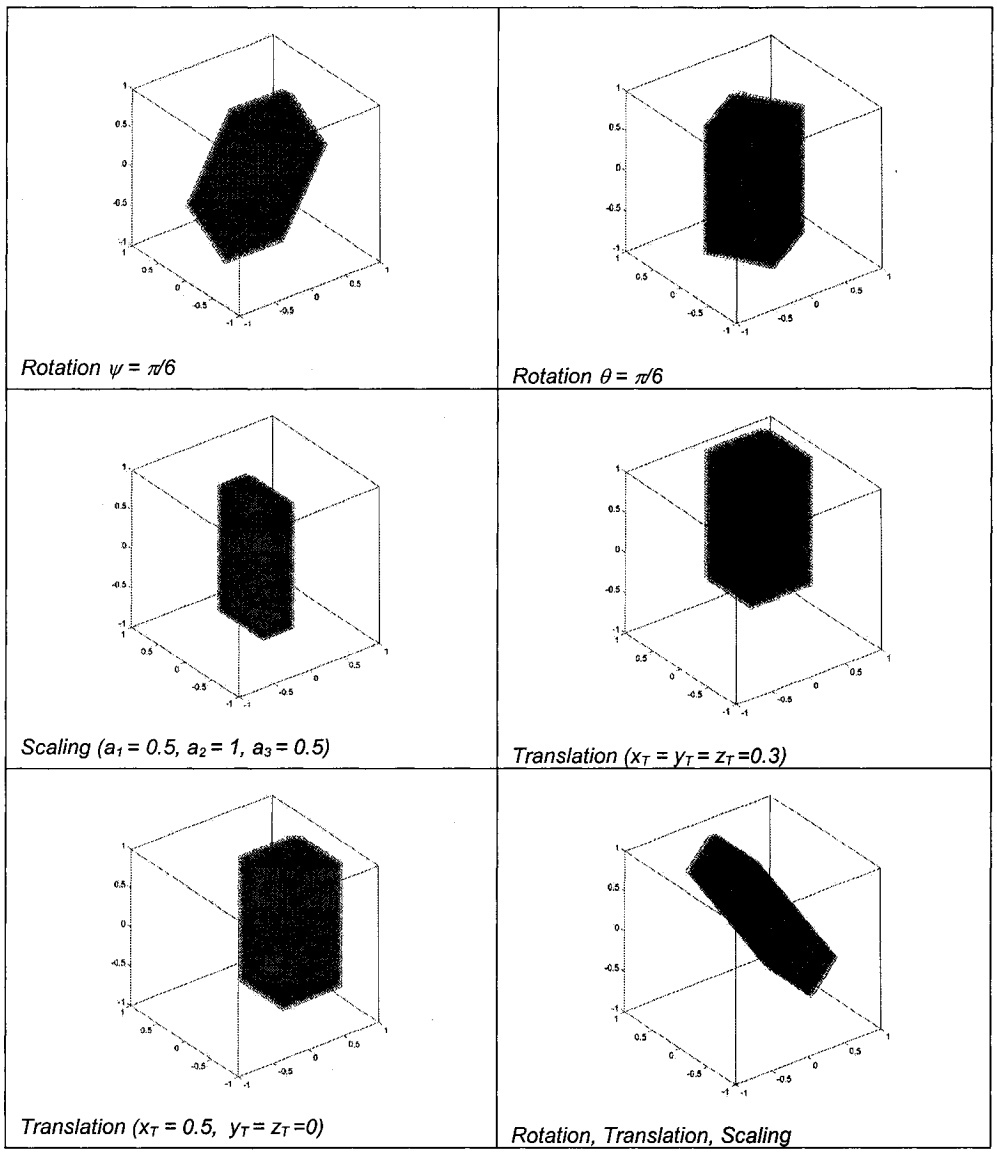
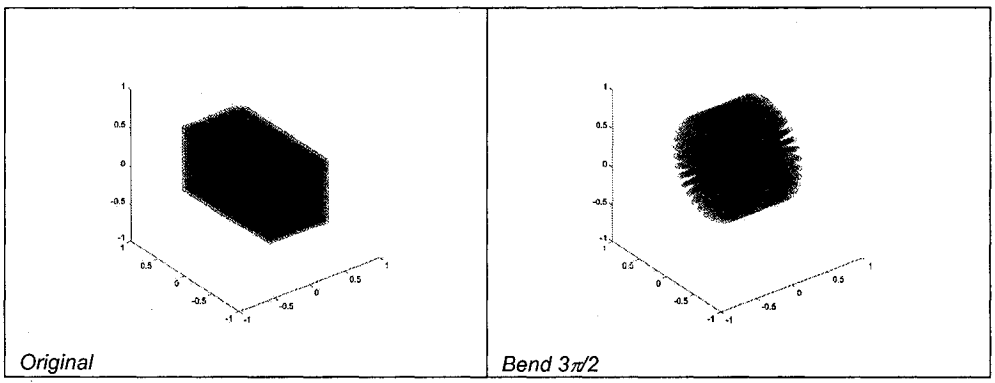


Figure 5.8. Affine transformations (generation mode)



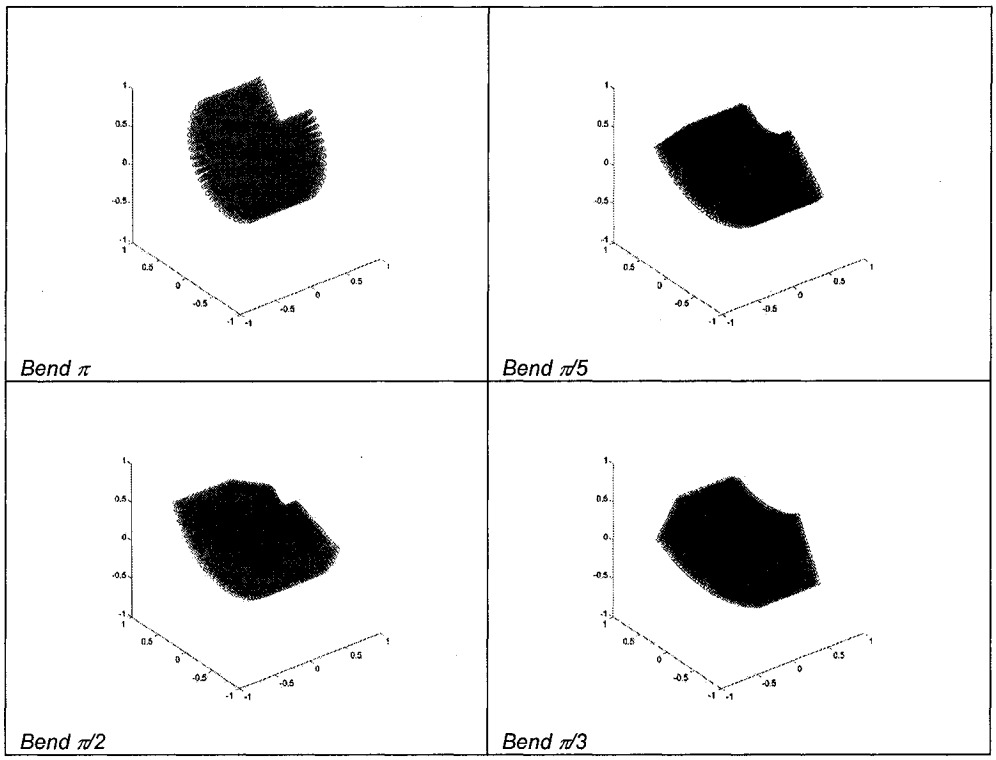
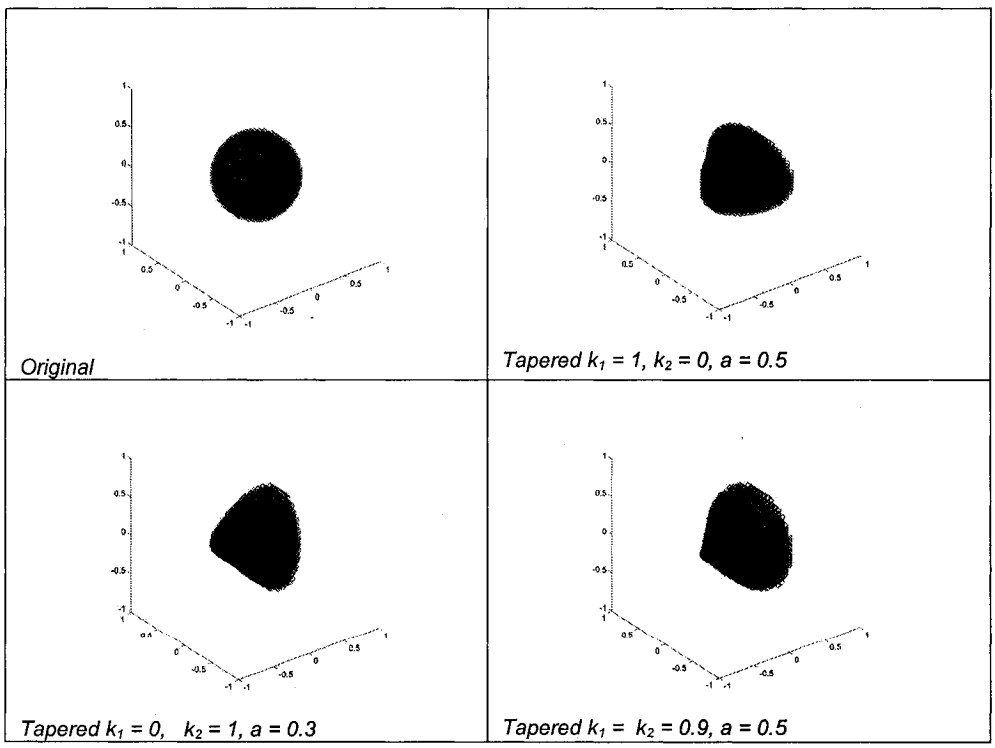


Figure 5.9. Bending deformation (synthetic data)



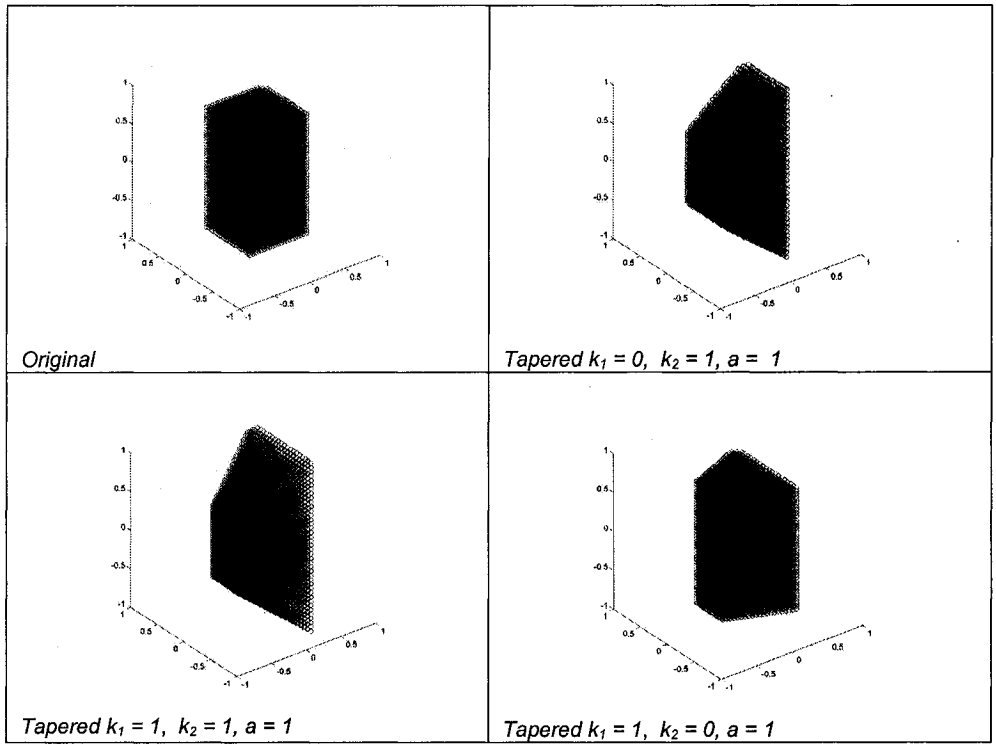


Figure 5.10. Tapering deformation

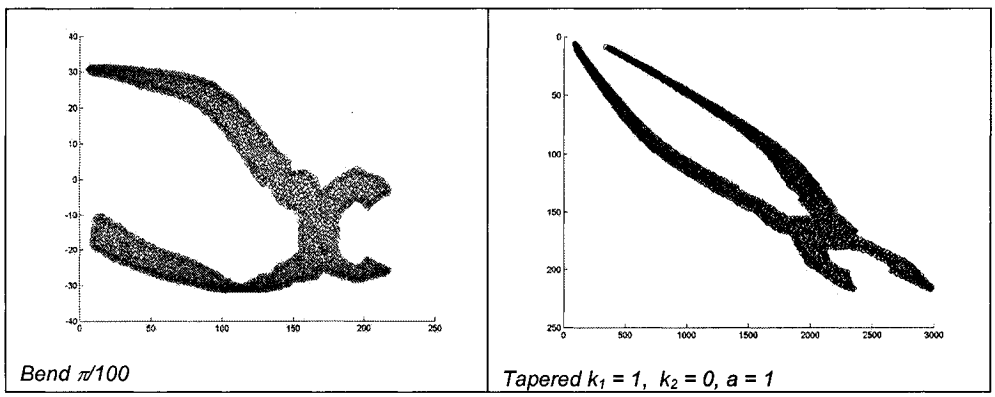
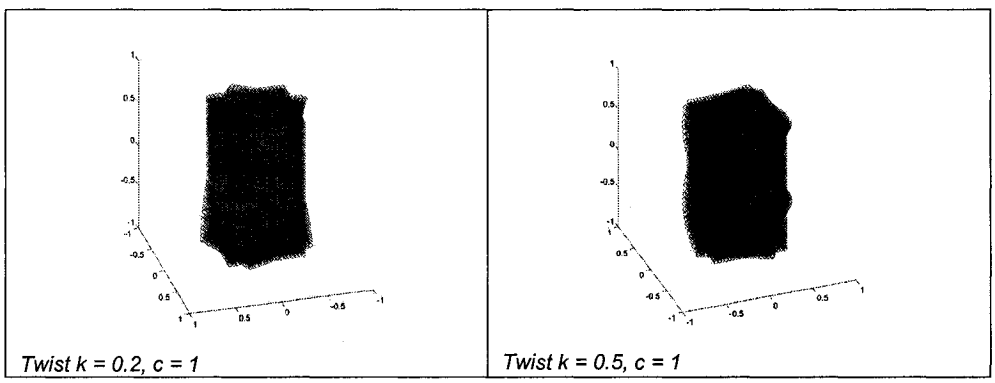


Figure 5.11. Bending and tapering deformation (range data)



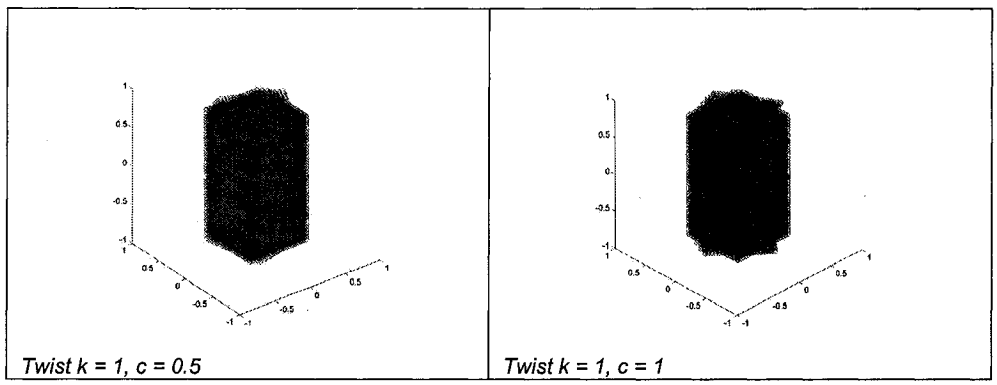


Figure 5.12. Twisting deformation

5.1.2.2. Training Mode

When provided with corresponding points from a given reference model as inputs and those of a transformed model as targets, the transformation module can learn and store in its weights information on how these points have been transformed and/or deformed. This information can be used for object classification and motion estimation, as shown in sections 5.1.3.4 and 5.1.3.5.

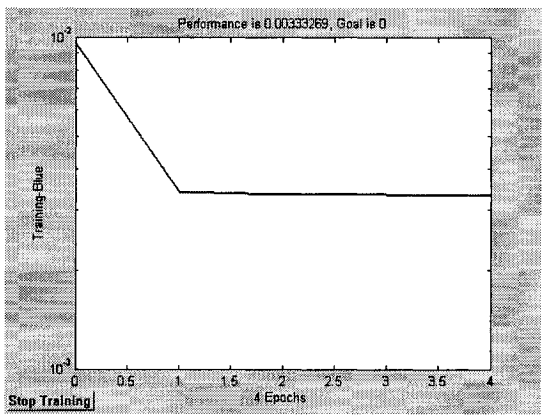


Figure 5.13. Learning diagram for training mode of transformation module

As in the case of the representation module, a normalization operation is performed on all the points of a given object such that they are scaled in the [–1 1] interval. Due to the fact that the module is used as complement to the representation module, and thus it is more a passive component, no other supplementary data are used in training mode. Scaled conjugate gradient backpropagation is used to train the network. Weights and biases are initialized using Nguyen-Widrow method. The parameter σ to approximate the second order information is set to a very low value $5 \cdot 10^{-5}$, and the same for the λ term that is added to make the Hessian positive ($5 \cdot 10^{-7}$), as for the training mode of the representation module.

The transformation module needs only few training epochs and a short computing time to efficiently recuperate the transformation parameters (25 sec/model).

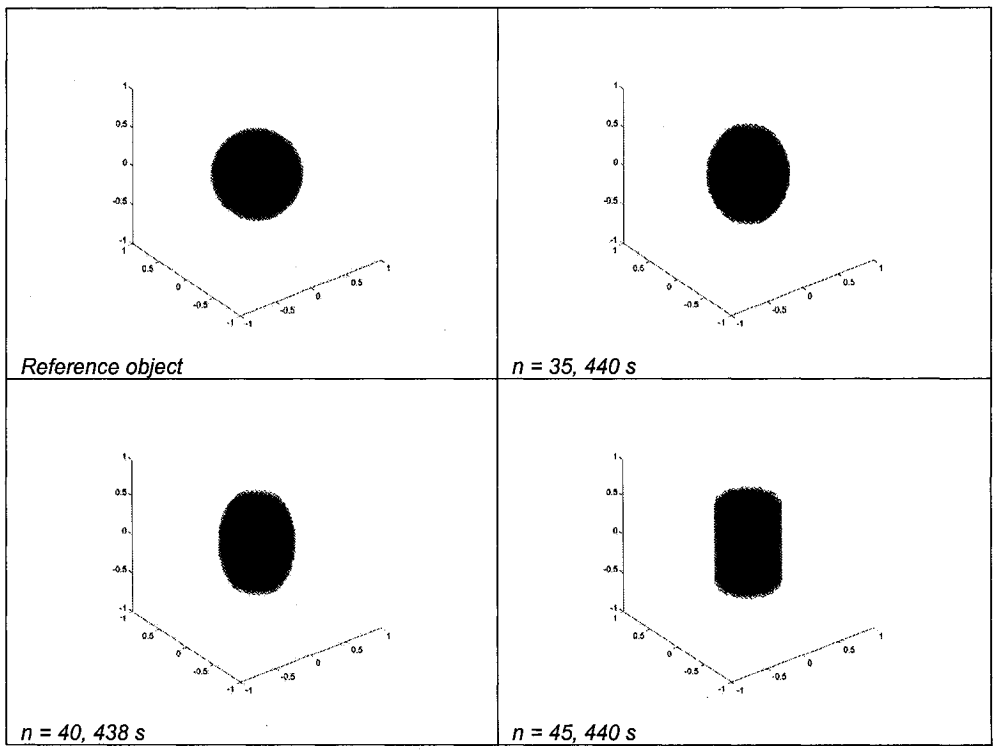
5.1.3. Applications

As described in chapter 3, the representation module can be used to perform object morphing, set operation and detect collisions between objects. Cascaded with the transformation module, more complex operations like object recognition and object motion estimation can be performed.

5.1.3.1 Object Morphing

In order to obtain object morphs, a linear interpolation is performed between the weight matrices of the two neural representations of the implied objects. For each step, the morphed object is obtained using as inputs the points of the [-1 1 –1 1 –1 1] object space and plotting those whose output result is less or equal to 0. Two conditions have to be fulfilled on the reference and target models in order to perform object morphing: the two objects must be aligned and they must be modeled using the same architecture to ensure the respective weight matrices have the same size. It takes approximately 7 minutes to build a morphed object.

Fig.5.14 shows the morphing operation of a sphere into a cone using $n = 50$ steps. Only a few steps are presented.



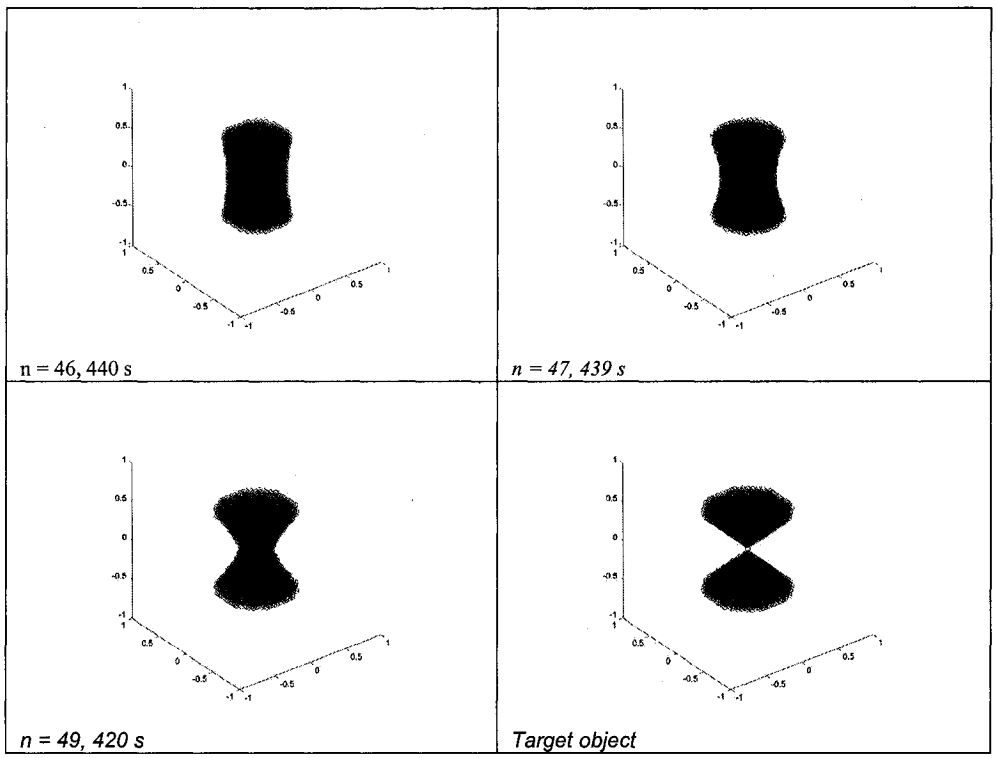


Figure 5.14. Morphing of a sphere into a cone

5.1.3.2. Set Operations

Using the property that, given the coordinates of a 3D point, the trained model of each object replies with a negative value if the point is inside the object or with a positive value if the point is outside, a point will belong to the union of two objects if it is inside the first object or inside the second object. In a similar manner, a point will belong to the intersection if it is inside the first object and inside the second object. A point will belong to the difference of two objects if it is inside the first object and outside of the second (or outside the first and inside the second).

The testing is performed for two spheres of equal radius 0.3, one centred at (0, 0, 0) and one centred at (0.3, 0, 0). Points from the [–1 1 –1 1 –1 1] object space are evaluated with the two neural networks corresponding to the two objects, and an "AND" or "OR" operation is performed between the outputs to determine if the points belong or not to the union/difference/intersection of the two objects. To assess visually the models resulting from set operations, only those points that belong to the desired operation are plotted (Fig.5.15). It takes approximately 15 minutes to build the model of a set operation.

Modeling of set operations does not require any conditions for the two objects, as long as their neural network representation exists.

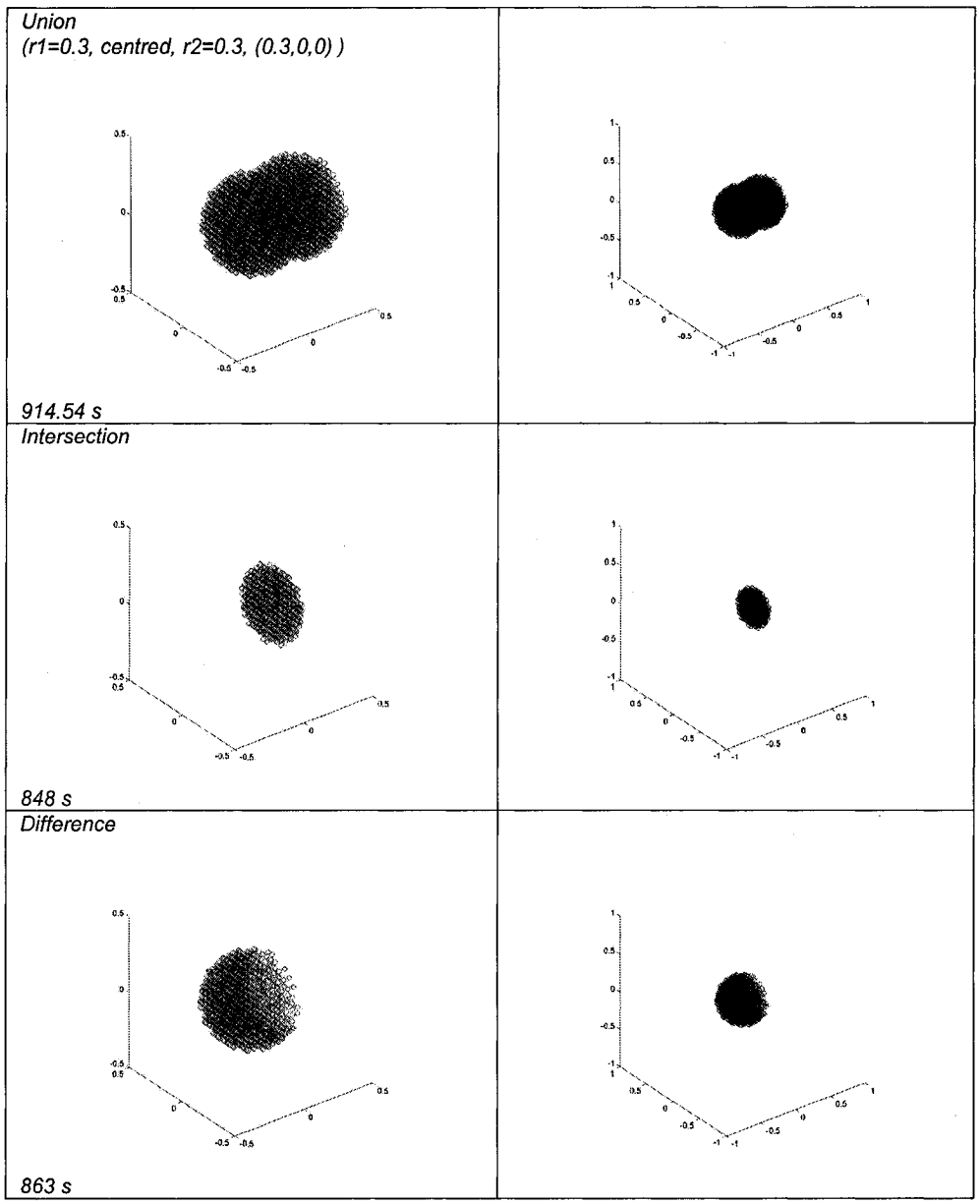


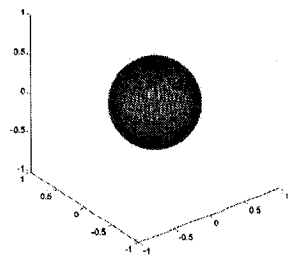
Figure 5.15. Set operations

5.1.3.3. Object Collision

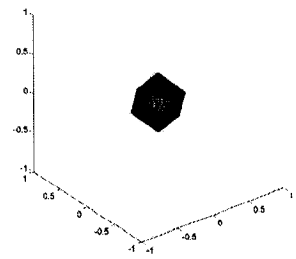
In order to detect collisions between a reference and a tested object, a set of points is sampled from the surface of the tested object and is substituted in the neural network model of the reference object. If the output results are negative, it means the points of the tested object are inside of the reference object and thus a collision occurred. The only requirement is that the neural representation of the reference object exists.

The tests are run with/without influence of noise for six scenes, as depicted in Fig.5.16. The first scene contains an object that is completely contained in the reference object. The object

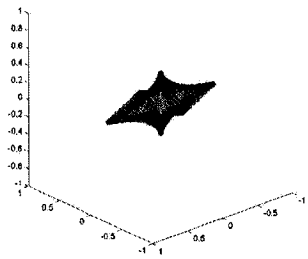
in the second scene has some points outside of the reference object, while the object in the third scene contains the reference model. The fourth test scene contains an object in collision with the model. The objects in the fifth scene are not in collision, while the ones in the last scene barely touch each other.



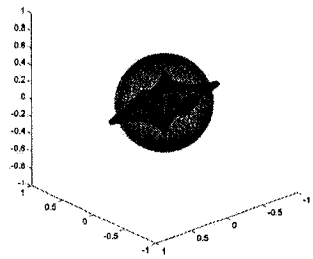
Reference object



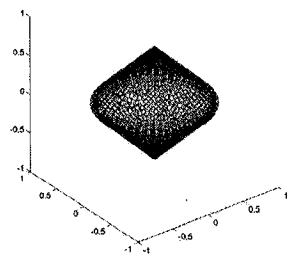
Test scene 1 (object inside)



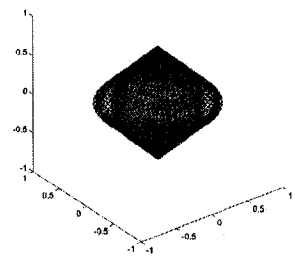
Test scene 2



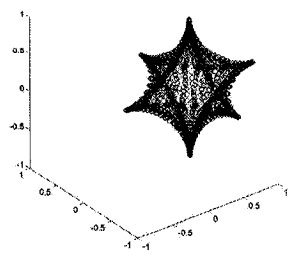
Intersection with the test model



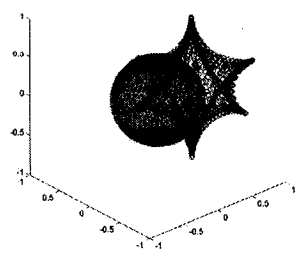
Test scene 3



Intersection with the test model



Test scene 4



Intersection with the test model

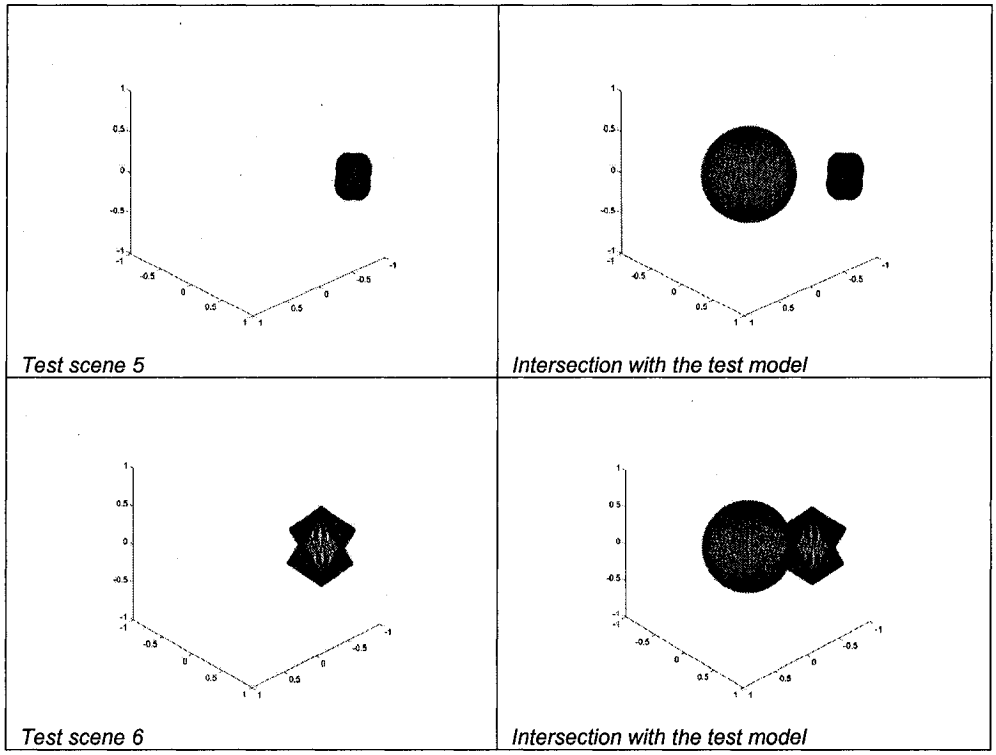


Figure 5.16. Testing scenes for collision detection

The results of the detection tests are presented in Table 5.2 for a number of 1200 sample points.

Table 5.2. Collision detection in test scenes

Percent of points detected inside the reference model	*No noise*	*Random noise level between 0 – 0.04*	*Random noise level between 0 – 0.05*	*Random noise 0 – 0.1*
Test scene 1	100%	99.5%	95%	58%
Test scene 2	96.56%	92.4%	88.7%	50%
Test scene 3	0%	0.2%	1%	0.9%
Test scene 4	42.8%	40%	38%	37%
Test scene 5	0%	0%	0 %	0%
Test scene 6	2.3%	1.8%	0.15%	2%

In case of non-noisy data, the results of detection are correct apart from the test scene 3 where the reference object is included in the test object. This situation occurs very rarely, but a simple way to detect this sort of collision is to repeat the procedure for testing if all the points of the sampled object are outside the reference object. Another option would be to use a set of sampled points from the object volume, not only from the object's surface. In this case, the above-mentioned problem is completely avoided. In most cases, the collision can be detected even in case of random noise. Problems appear in the case of barely touching objects. Because of

the random error, the user is no longer able to decide if the objects are collided or not (case of noise 0 – 0.05, scene 6).

The number of sampled points has also an influence on the recognition rate as depicted in Table 5.3. The more sample points used, the biggest the chance is that the collisions are detected correctly. However, this comes at the cost of computational time. For too many sample points, the detection can no longer be done in real-time.

Table 5.3. Influence of the number of sample points over collision detection time

Number of sample points	*Detection time (s)*
18	0.79
60	0.99
1200	1.8
1600	2.3
5000	21
10000	124

Due to the fact the representation given by the neural network is also a volumetric one, a situation of object collision can be also detected using set operations. If the result of the intersection operation between the two objects is the empty set, then it means that no collision occurred. However, this implies the existence of neural models for both implied objects and the fact that the set operation is very accurate (which is not always the case, especially in the presence of noise).

5.1.3.4. Object Classification and Recognition

Object recognition refers to the process of identifying an unknown (transformed – rotated, translated, scaled, bent, tapered, twisted) object from a given model database. In order to perform this task, a neural network representation of all the objects must exist and must be stored in a database of reference models.

The model-based matching is done by first aligning the transformed object with the reference model in the database using the transformation module. The neural network of the transformation module is trained using the reference model points as training points and the points of the transformed model as targets. In this way, the transformation information is stored the weights of the network. After the alignment procedure, a set of points is sampled from the transformed object surface and the points are tested if they belong or not to the reference model.

Four scenes (Fig.5.18) containing synthetic data of 4 (non-noisy) objects (Fig.5.17) are used during tests. The representations of these objects in the database are built using inputs X, Y, Z and X^2, Y^2, Z^2, with 2 extrasurfaces situated at 0.05 from the objects' surfaces, and using the same number of neurons in the first and second hidden layers 7-3.

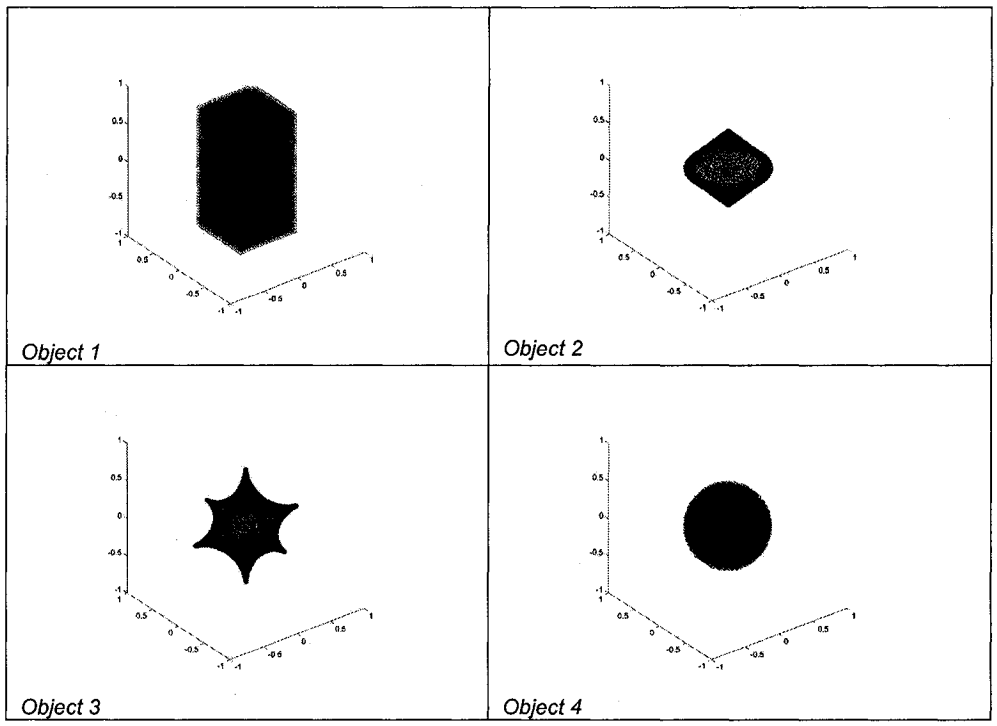


Figure 5.17. Neural network representation of objects in the model database

The objects are transformed randomly (translated, rotated, scaled, bent or tapered in the given example). Tests are run for different levels of noise and different number of samples.

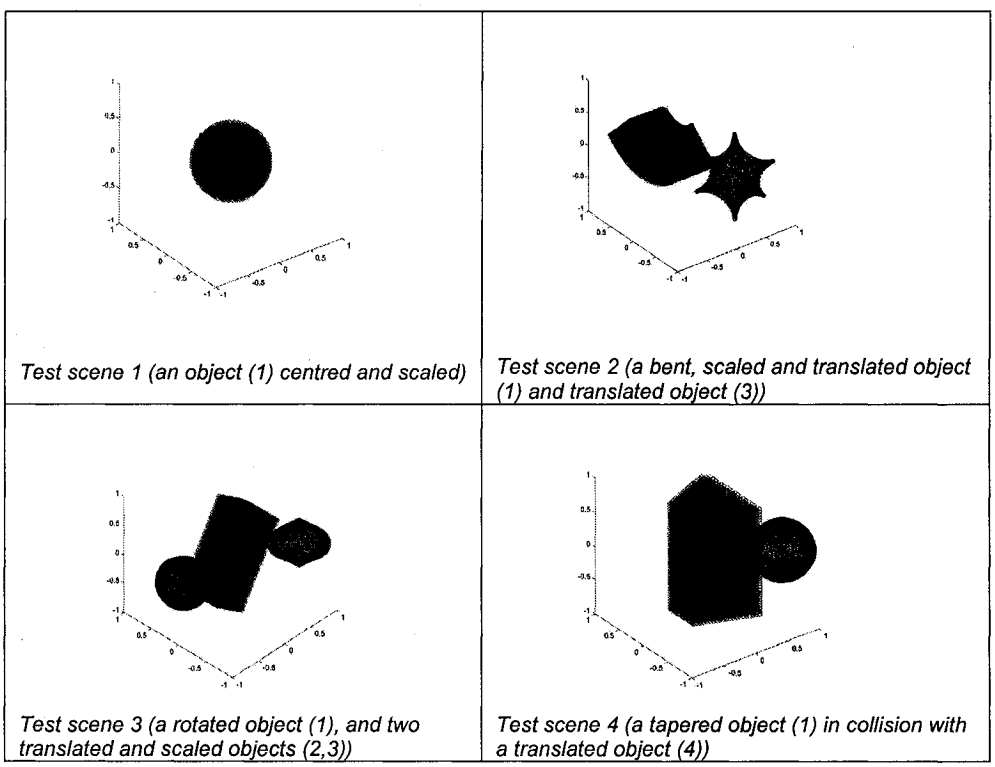
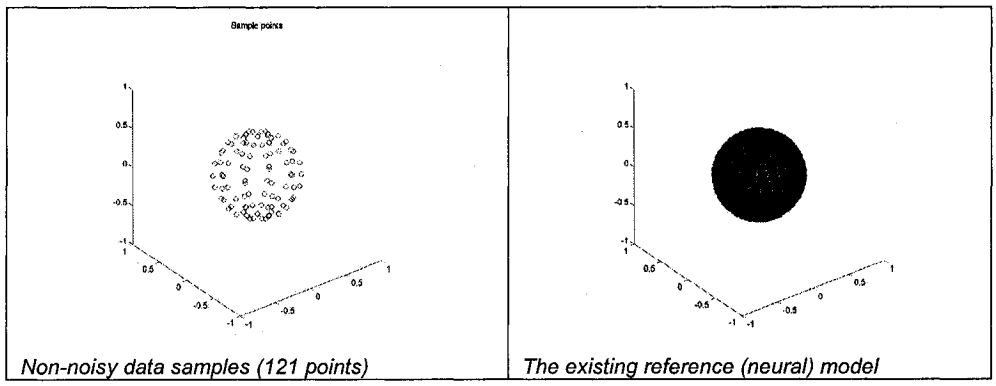


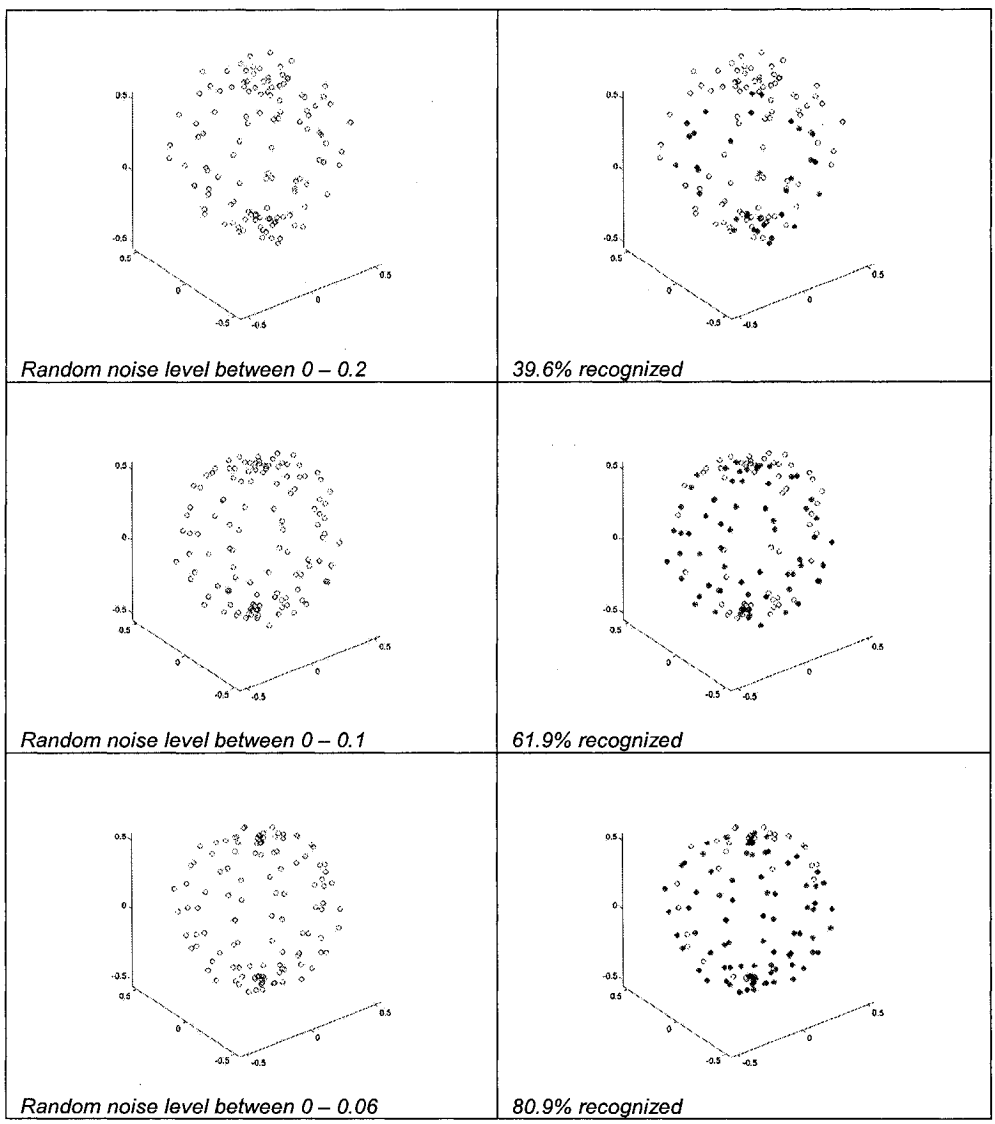
Figure 5.18. Recognition testing scenes

In case of a simple scene (first scene from Fig. 5.18), the neural model is able to recognize 100% of the points in case of a tolerable level of random noise (<0.04), as depicted in Fig. 5.20.



Non-noisy data samples (121 points) | *The existing reference (neural) model*

Figure 5.19. Non-noisy data samples and the neural model for test scene 1



Random noise level between 0 – 0.2 | *39.6% recognized*

Random noise level between 0 – 0.1 | *61.9% recognized*

Random noise level between 0 – 0.06 | *80.9% recognized*

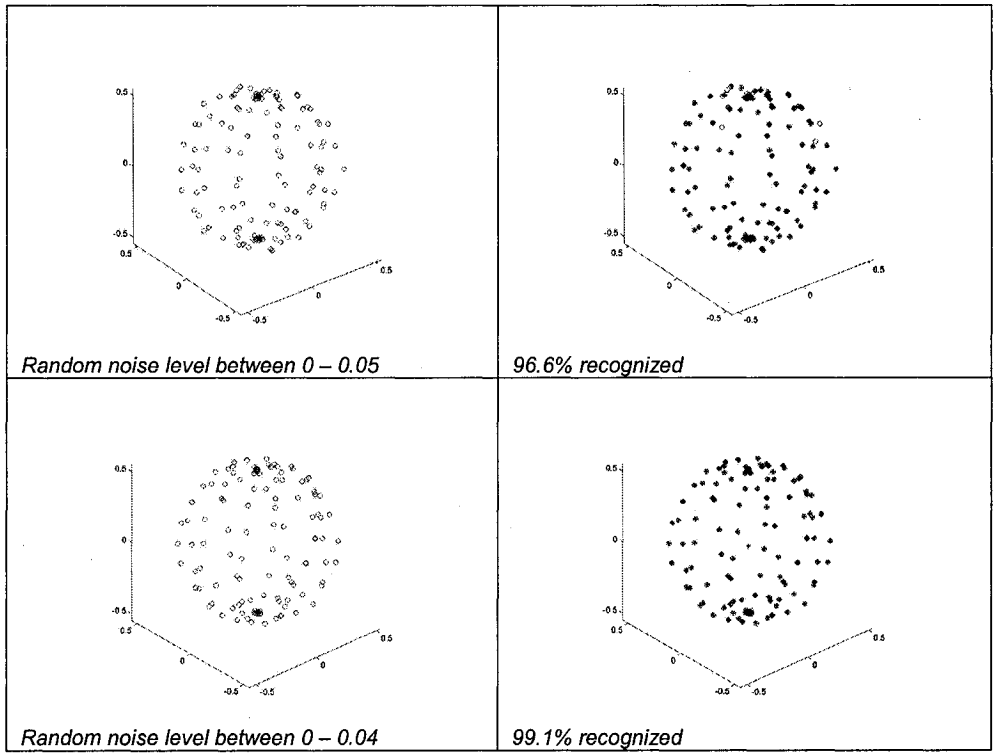


Figure 5.20. Influence of different levels of random noise over the recognition rate

Only 60 percent of the points are recognized for a random noise level between 0 and 0.1 and the network is no longer able to recognize an object corrupted with noise between 0 – 0.2. The recognition rate is not influenced by the number of sample points used for matching, as demonstrated in Table 5.4.

Table 5.4. Influence of random noise and of the number of sample points over the recognition rate

No of points/ Random Noise level	*121 sample points*		*441 sample points*		*961 sample points*		*1681 sample points*	
	Recognized		*Recognized*		*Recognized*		*Recognized*	
	No of. Pts.	*Percent*	*No of. Pts.*	*Percent*	*No of. Pts.*	*Percent*	*No of. Pts.*	*Percent*
0-0.2	48	39.6%	135	30.6%	289	30%	554	32.9%
0-0.1	75	61.9%	257	58%	550	61%	971	57.7%
0-0.06	98	80.9%	351	79.5%	758	84.1%	1305	77.6%
0-0.05	117	96.6%	419	95%	915	95.2%	1605	95.4%
0-0.04	120	99.1%	440	99.7%	959	99.7%	1677	99.7%
<0.04	121	100%	441	100%	961	100%	1681	100%

In case of a complex scene, with many objects, the segmentation of the scene is required prior to the recognition process. The k-means clustering algorithm is used to divide the scene in the component objects (Fig.5.21). For the testing scenes where objects are not collided, between 0.5% and 5% of the points are misclassified due to errors in the clustering algorithm. A higher number of misclassified points occurs in the test scene 4, where the objects are collided.

After the clusters are built, each resulting object is aligned using the transformation module to the objects in the database.

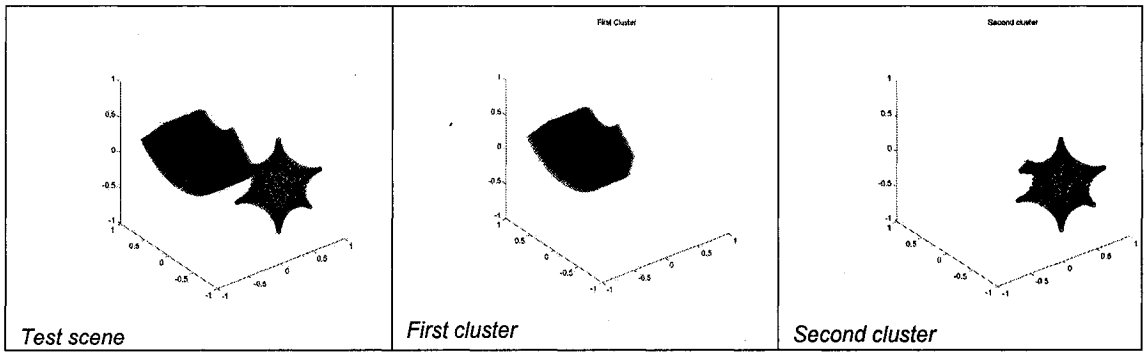


Figure 5.21. Resulting clusters for the second scenario (1% of points is misclassified)

Fig. 5.22 shows the influence of the noise level on the alignment process in the third scene, as depicted in Fig. 5.18. For a reasonable amount of noise, the transformation module is able to align correctly the model to the transformed object. However, for noise up to 0.2 the alignment procedure is quite badly affected.

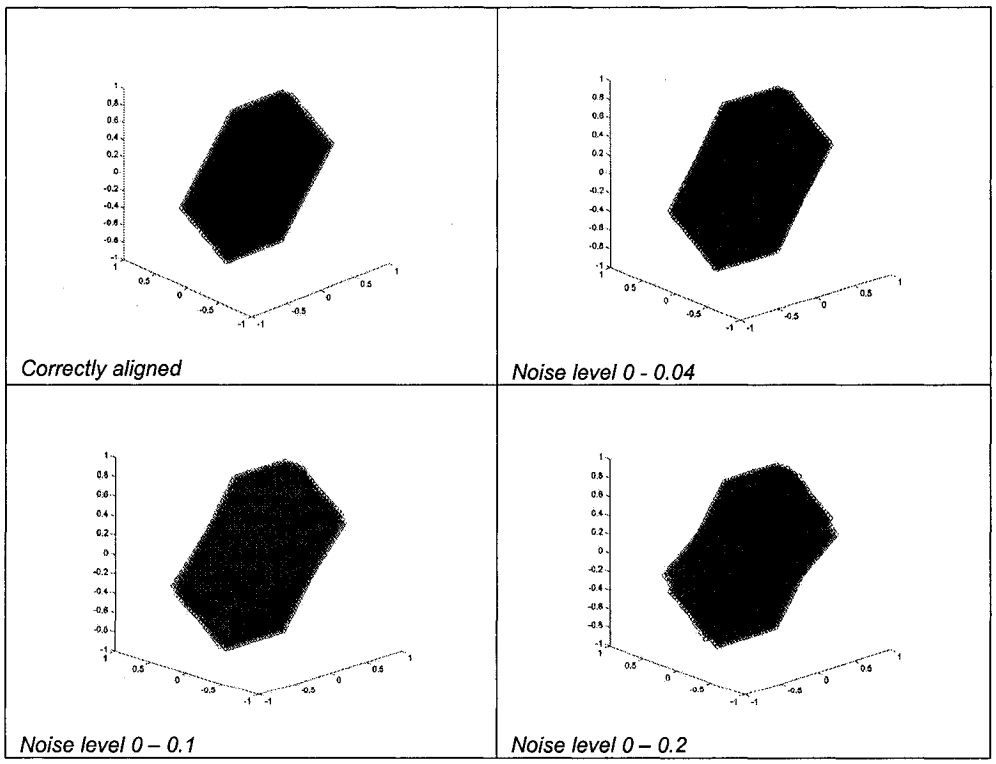


Figure 5.22. True and estimated transformed object for the third scenario

The aligned objects are then compared for matching as shown for the first scenario. A set of 121 randomly sampled points added with different levels of random noise (from 0.01 to 0.2) is

used to check the model matching. Table 5.5 shows the average percentage of points recognized in each scene as belonging to one of the object models for the case in which a random noise between 0 and 0.04 is added to the sampled points. All objects are correctly detected in the four scenes from Fig. 5.18.

Table 5.5. Percentage of recognized points belonging to models in the database for each scenario (case of a noise level 0-0.04)

Recognized points	*Object 1*	*Object 2*	*Object 3*	*Object 4*
Scene 1	2.4%	4.9%	1.6%	99.1%
Scene 2	93.3%	3.3%	91.7%	3.1%
Scene 3	95%	99.1%	5.78%	98.3%
Scene 4	91.7%	6.6%	1.6%	92.5%

The percentage of recognition is high in the scenes where objects are not collided and objects are not bent or tapered. The recognition is harder for objects with sharp edges in the case they are rotated (the star in scene 2, for example).

5.1.3.5. Object Motion Estimation

Given a known 3D object already represented by a MLFF neural network, the transformation module can be employed to estimate the motion based on a sequence of sets of 3D range data taken while the object is moving.

The motion of a 3D object is described in terms of rotation, translation and scaling, as given in the TRPY generalized motion matrix. The reference object is assumed centred in the origin, and an initial guess is made for all the implied parameters: the translation and rotation parameters are initially set to 0, and the scaling factor to 1 (the size is supposed not to change). The motion parameters are computed using the transformation module at each time step. As the motion and the measurements are considered continuous, the next translation parameter is always initialized with the position of the object's centroid in the previous step. The rotation angles are set also to their previous value (as it is unlikely that abrupt changes occurred since the last movement step) and the scaling factor to 1. The procedure is repeated until there are no more movements in the scene. It is considered that no more movements have been made when for four consecutive steps there has been no change in any of the parameters.

First, only the network capability to detect translation parameters under different levels of noise is studied. A spherical object is considered to be moving along a 3D trajectory as in Fig.5.23. The movement is also decomposed along each direction to see the estimation for each case.

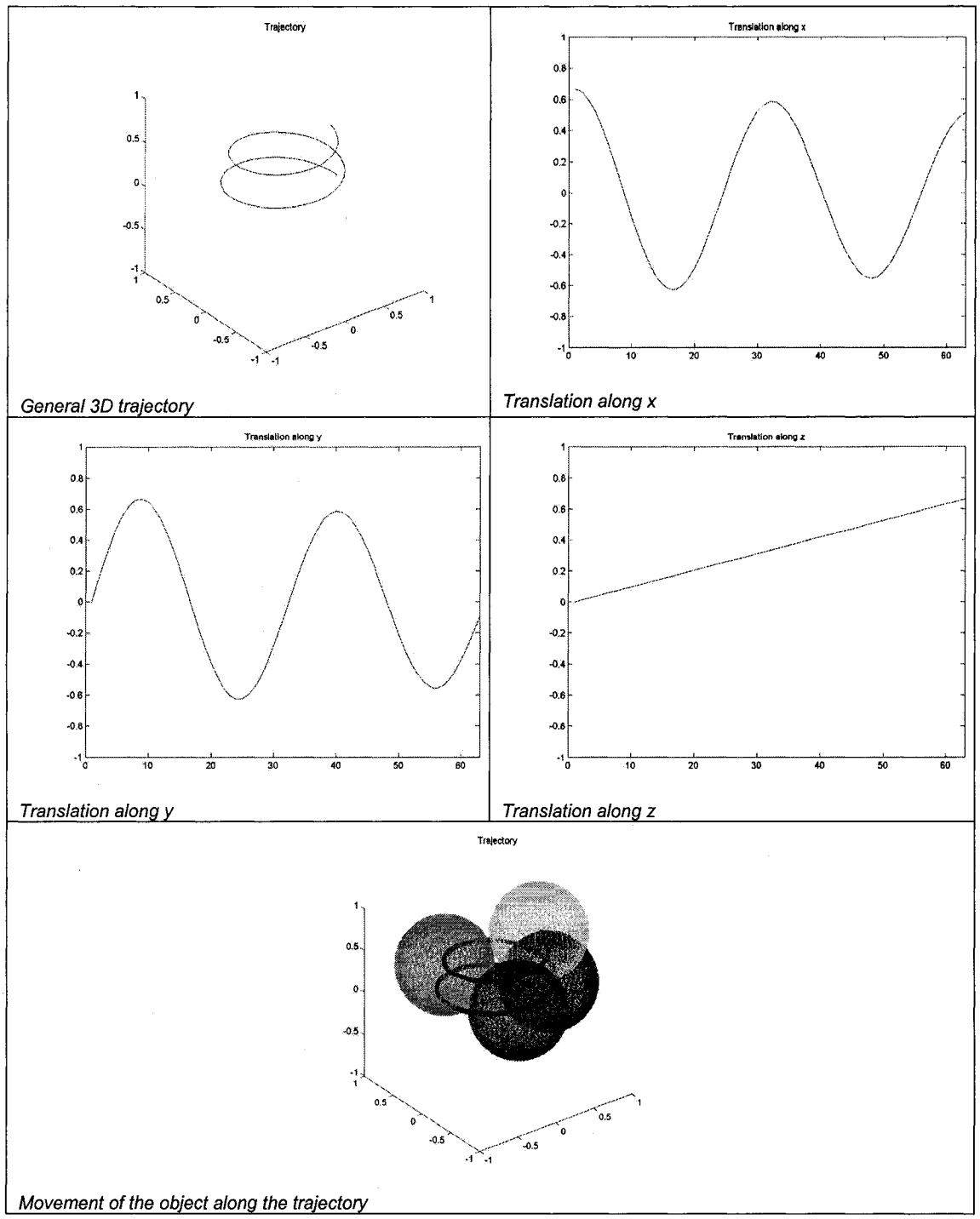
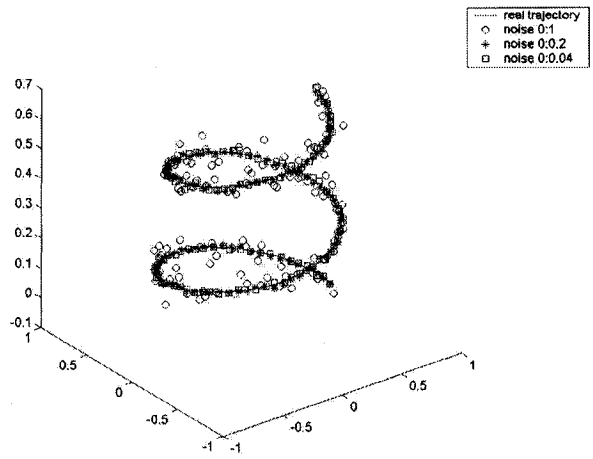
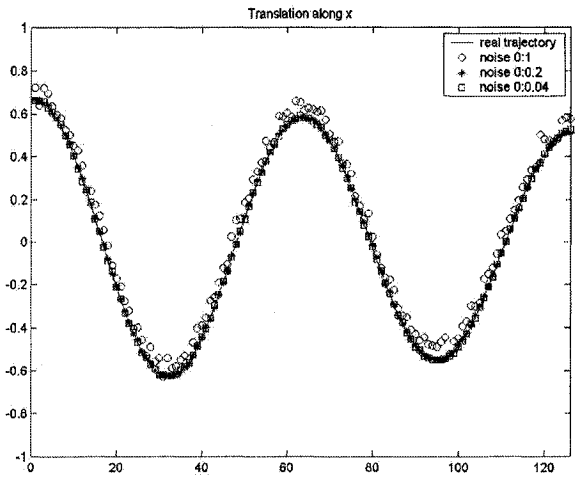


Figure 5.23. Framework for translation estimation

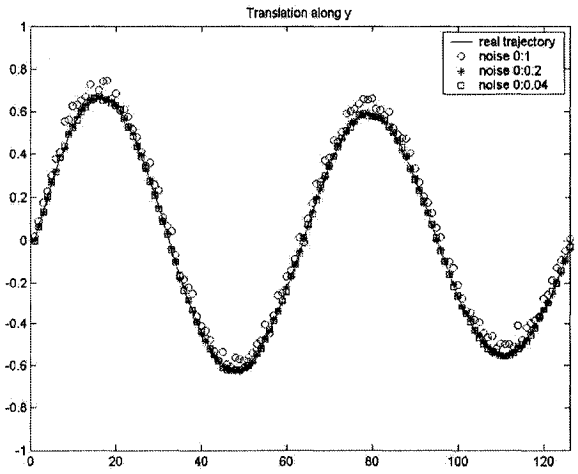
Tests are run for 60 time steps over which the spherical object moves from one end to the other end of the trajectory, for different levels of noise. The results are presented in Fig. 5.24.



Real and estimated general 3D trajectory



Real and estimated translation along x



Real and estimated translation along y

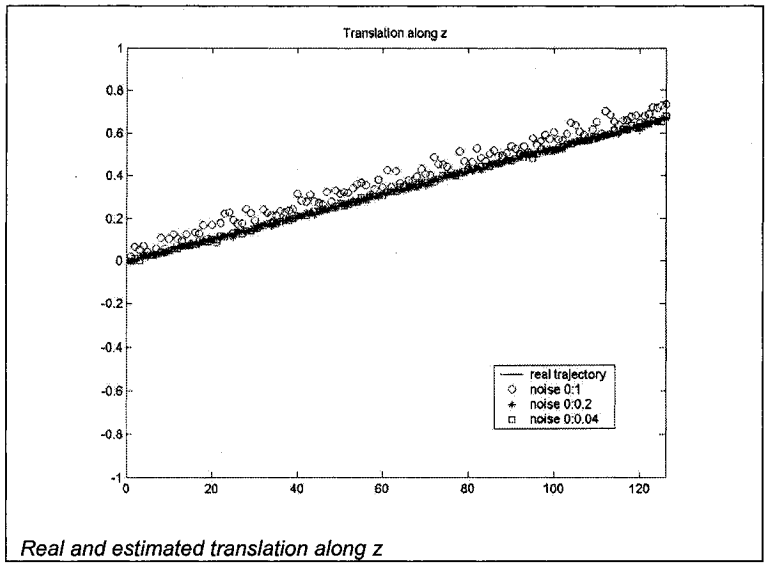
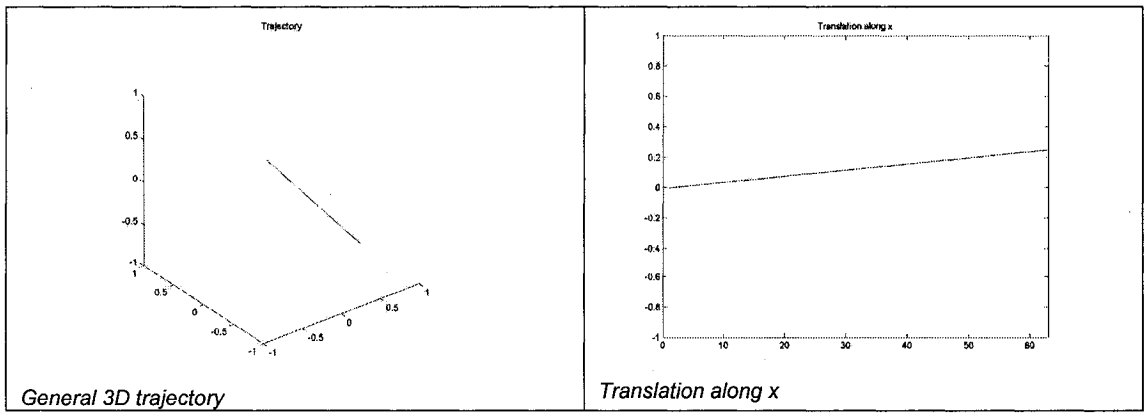


Figure 5.24. Translation estimation for different levels of random noise

The results show that the reconstruction of the trajectory is almost perfect for a reasonable noise level (0-0.2), but becomes very difficult for high levels of noise (0-1). However, even in this case, the reconstructed trajectory gives a hint on the real shape of the real trajectory. It can be easily deducted from the reconstructed trajectory that the movement is somehow sinusoidal along x and y, and probably a straight line along z.

In the second test scenario, the network is employed to estimate not only the translation components, but also the scaling and rotation. The testing framework is presented in Fig. 5.25. A cubic object is moved along a straight line in 3D, while it is also rotated many times and scaled once with a factor of 0.1 along all directions. There are 65 steps considered in the movement of the object.



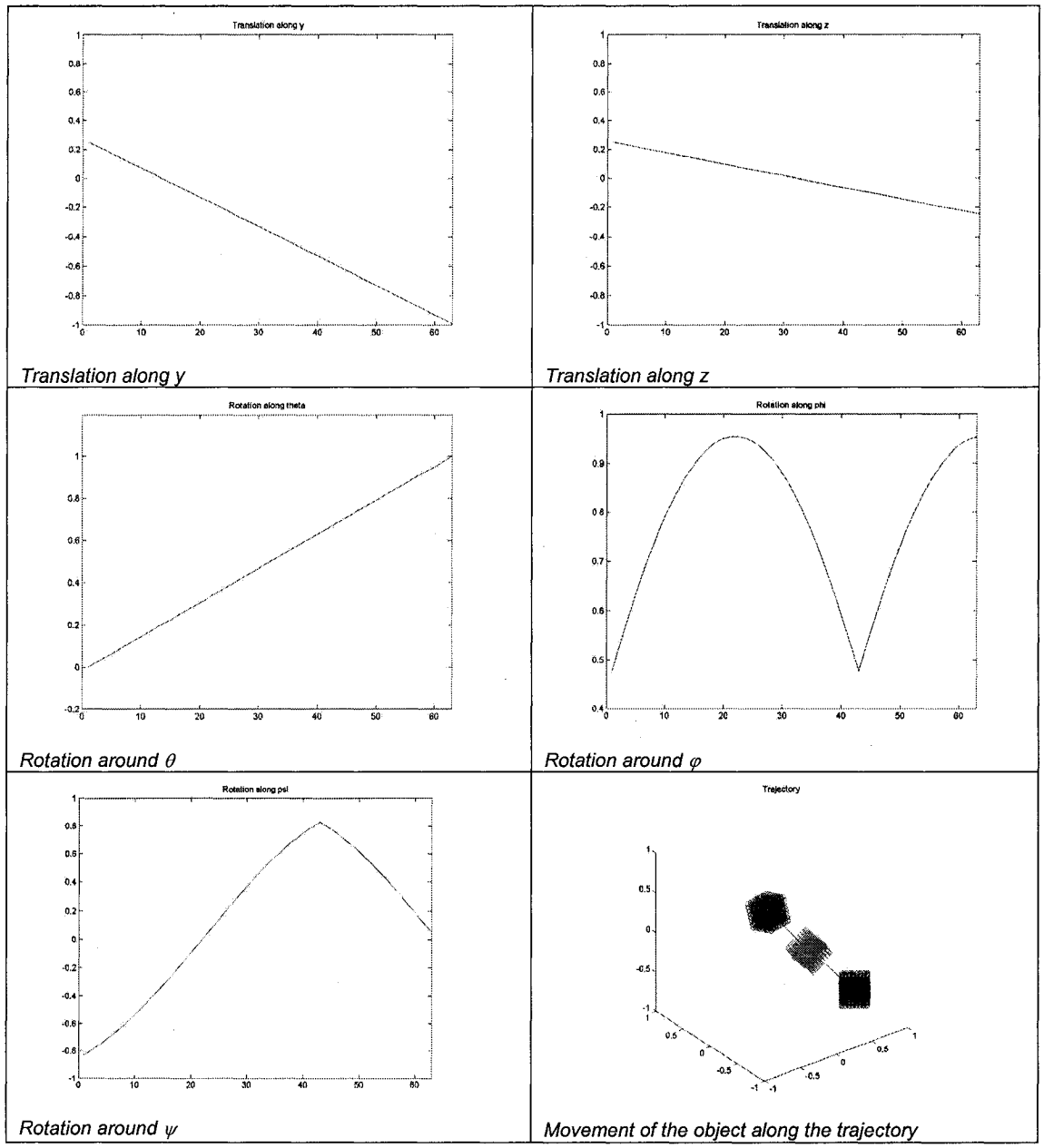
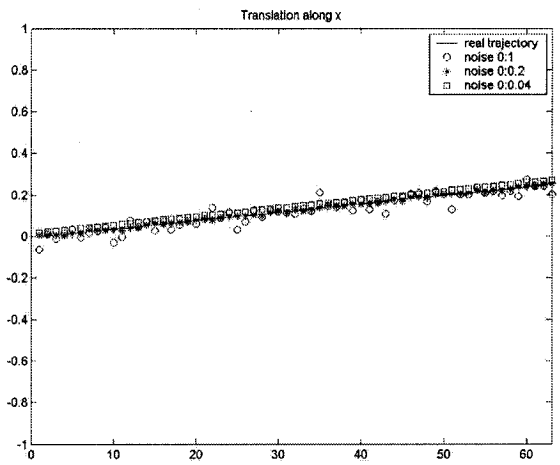


Figure 5.25. Framework for translation and rotation estimation

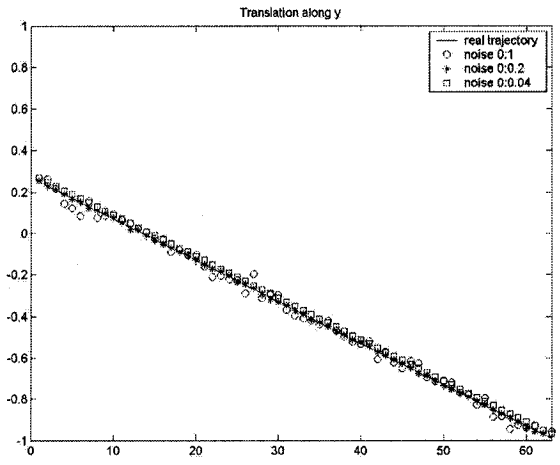
Table 5.6 shows the estimation of the scaling parameter along the x, y, and z axis and Fig. 5.26 shows the estimated trajectory and estimated angles using the transformation module. The presented results demonstrate that the prediction of rotation angles and scaling factors is correct, even in case of noise.

Table 5. 6. Estimation of scaling factor

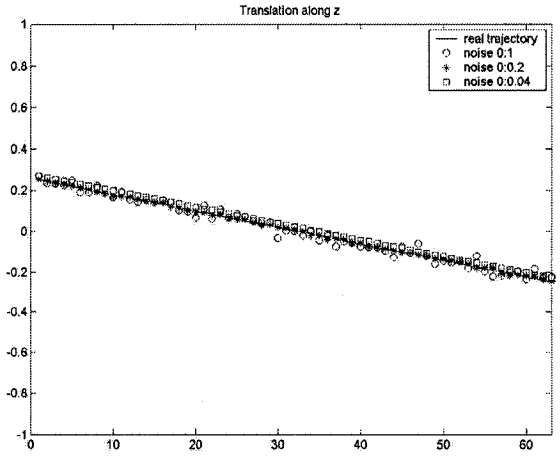
	Scaling along x	*Scaling along y*	*Scaling along z*
Real value	0.1	0.1	0.1
Noise 0-1	0.18	0.2	0.18
Noise 0 –0.2	0.12	0.12	0.115
Noise 0-0.04	0.11	0.1	0.1



Translation along x



Translation along y



Translation along z

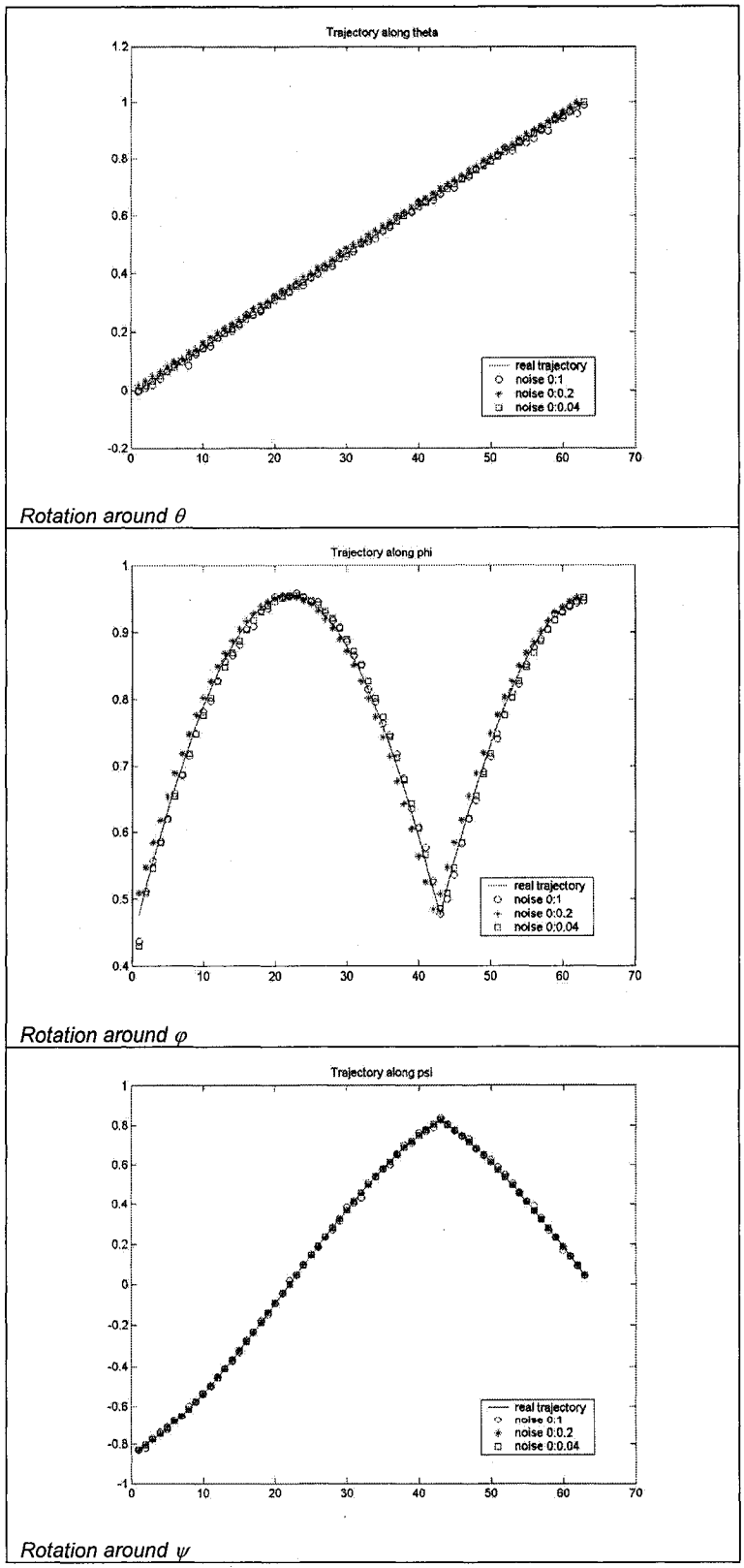


Figure 5.26. Translation and rotation estimation for different levels of random noise

5.2. Self-Organizing Network Models

5.2.1. Self-Organizing Map Evaluation

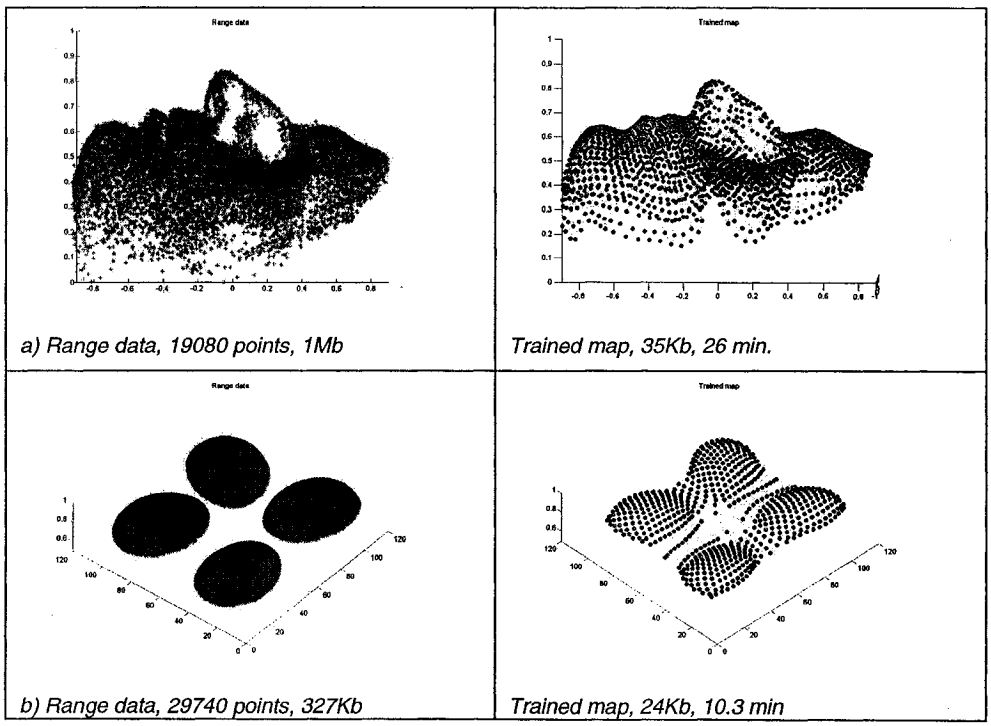
Data Sets

Range data from the NRC database and synthetic data are used to test the SOM network. Data is normalized such that it has 0 mean and variance of 1, as shown in section 4.4.

Learning and Parameter Adjustments

The weights are initialized linearly, along the two principal eigenvectors of the data set and a hexagonal lattice is used. The network is trained using the batch version of the training algorithm. The considered neighbourhood function is a Gaussian. Tests are conducted for different initial values of the learning rate and of the neighbourhood radius.

Fig.5.27 shows some models obtained by training the SOM network. It can be noticed first that the quality of the model is good as long as the modeled object does not contain cavities (Fig.5.27a). However, the SOM has problems for objects with cavities (Fig.5.27d) and it is not able to model entire scenes (Fig.5.27b, c) due to the fixed grid of neurons. The modeled objects can be stored with less memory usage. The 1Mb pointcloud of the face is compressed to a model of 35Kb. The modeling time is quite fast (section 5.3.3) in comparison with the MLFF neural network.



a) Range data, 19080 points, 1Mb

Trained map, 35Kb, 26 min.

b) Range data, 29740 points, 327Kb

Trained map, 24Kb, 10.3 min

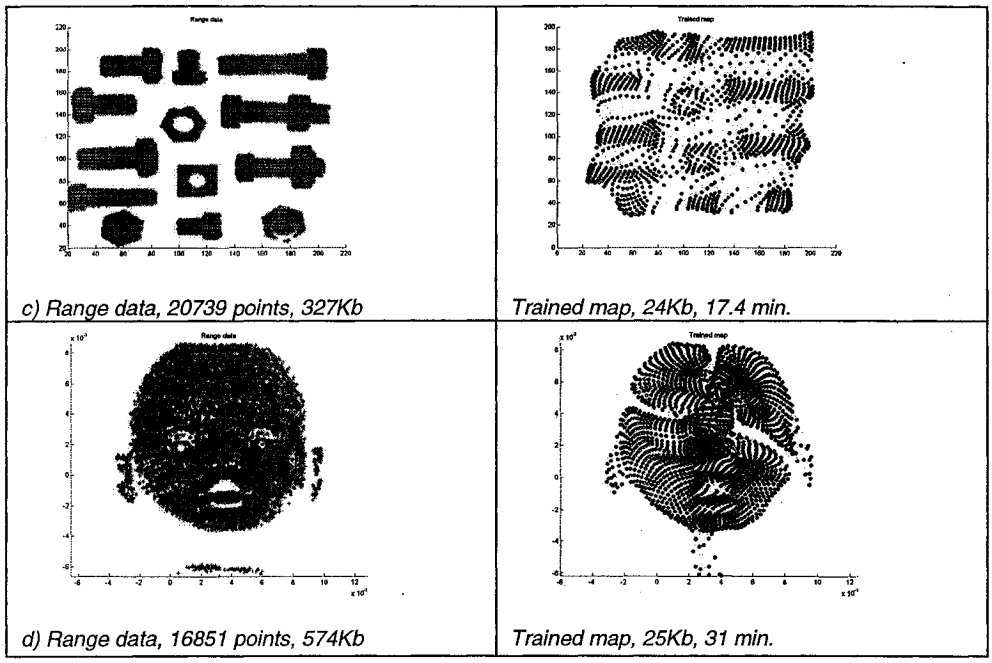


Figure 5.27. SOM models for a map size of 45×55 and 500 training epochs

As mentioned in chapter 4, section 4.5, in order to evaluate the quality of the models two simplistic measures of the resolution (*qe*) and topology preservation (*te*) are used. Fig. 5.28 shows these errors as a function of the number of training epochs for a fixed size map of 25×45 and a value of 5, respectively 1 for the starting value and final value of the radius of the neighbourhood ($\sigma_0 = 5$, $\sigma_T = 1$, in eq. (4.9), section 4.2.3.3).

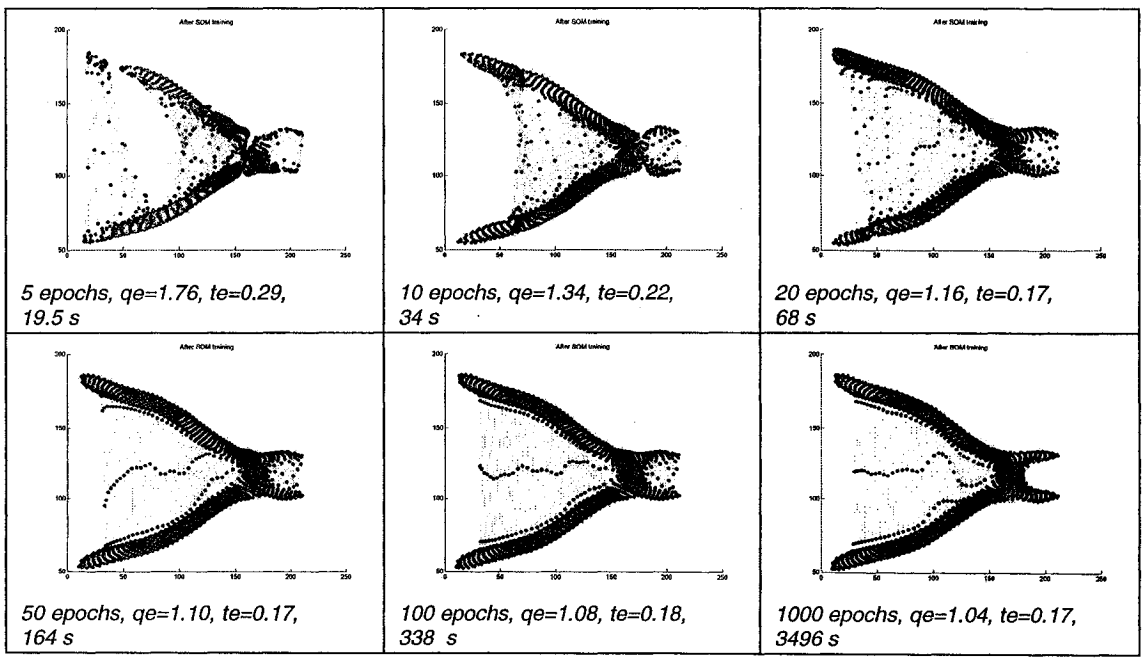


Figure 5.28. SOM errors for a fixed size map of 25×45

The computation time is constant and is approximately 3.4s/epoch. The graph in Fig 5.29 shows the trend of the quantization and topographic error with the number of training epochs. As expected, the quantization error decreases with the number of training epochs, showing the learning is improving, and therefore the quality of the model is better. The topographic error remains relatively constant, to a low value, indicating the SOM has a good topology preservation capacity.

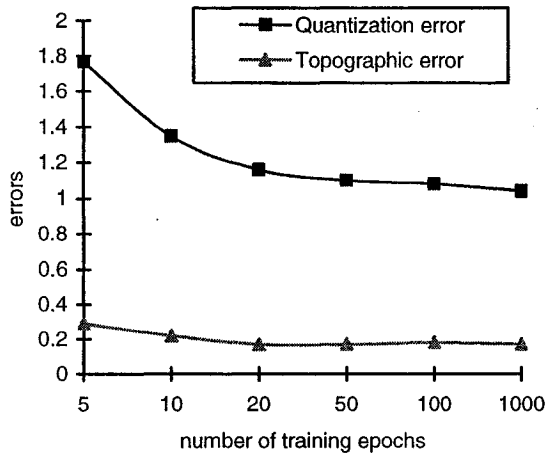


Figure 5.29. SOM evolution of quantization and topographic error with the number of training epochs

Another set of tests is conducted to show the influence of the neighbourhood function choice on the learning procedure. Fig. 5.30 shows the influence of the initial value of neighbourhood radius σ_0, for a fixed size map and a fixed number of training epochs over the quality of learning.

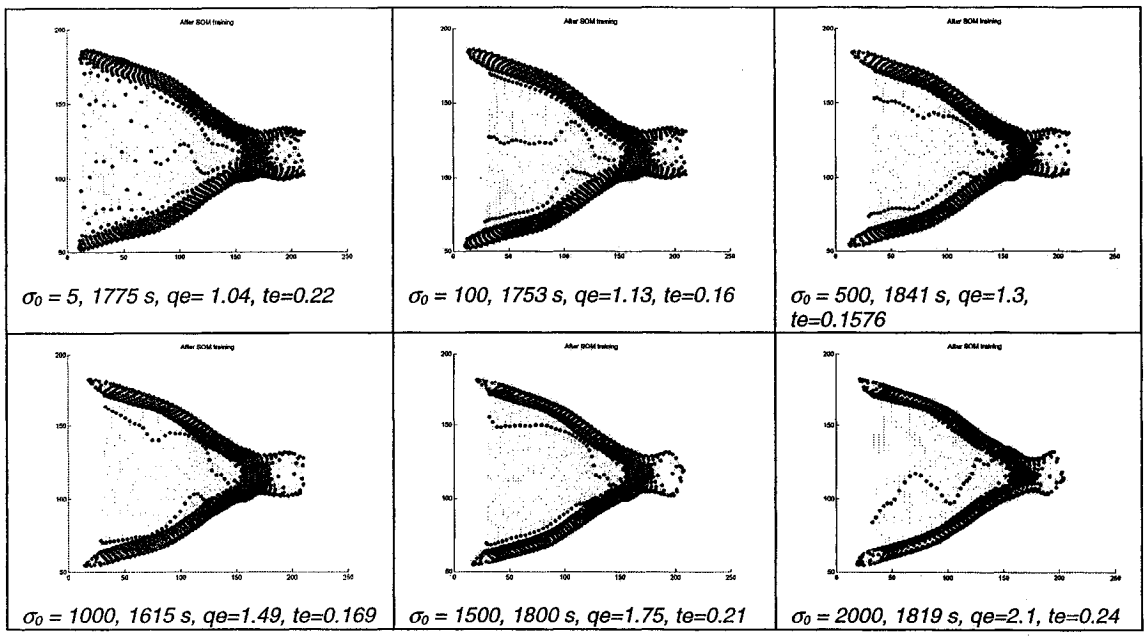


Figure 5.30. Influence of neighbourhood radius σ_0 for fixed map size of 25x45 and 500 training epochs

If the neighbourhood is too small, the spatial map cannot be organized properly and results in large errors. It is recommended that the initial value to be wide enough. However, a too large initial value leads to increases in the values of both quality parameters. As resulting from the tests we performed, the best compromise between the visual quality and the values of the errors is obtained for a value of σ_0 that is around half the dimension of the map size.

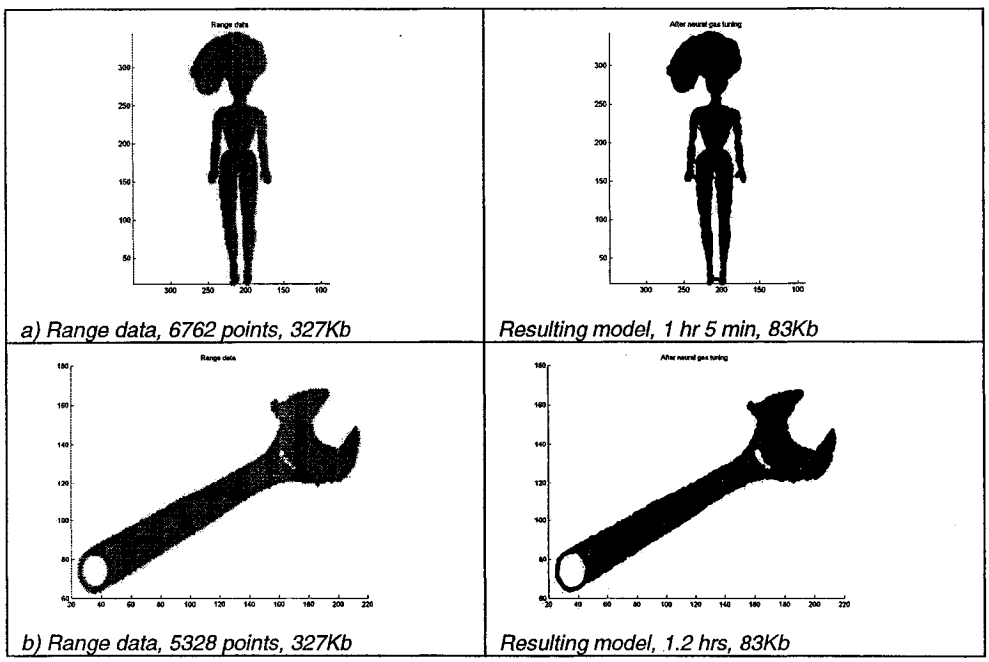
5.2.2. Neural Gas Evaluation

Data Sets

The same NRC range data set is used to train the neural gas network. Data is normalized such that it has 0 mean and variance of 1, as shown in section 4.4. Different models of different complexities are tested.

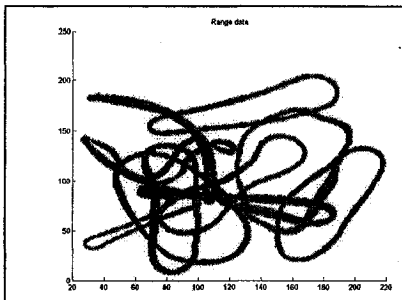
Learning and Parameter Adjustments

The weights are initialized randomly with a very small value, of the order 10^{-5}, such that symmetries in the data set are avoided. Tests are run for different initial values of the learning rate and the parameter λ that rules the ranking procedure. Fig. 5.31 shows some models of different complexity obtained with the neural gas network. The obtained models reproduce very well the training data.

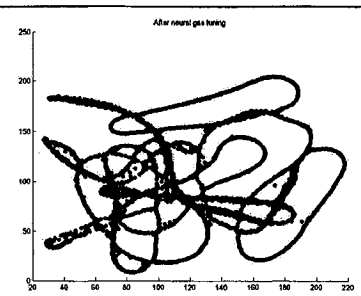


a) Range data, 6762 points, 327Kb | *Resulting model, 1 hr 5 min, 83Kb*

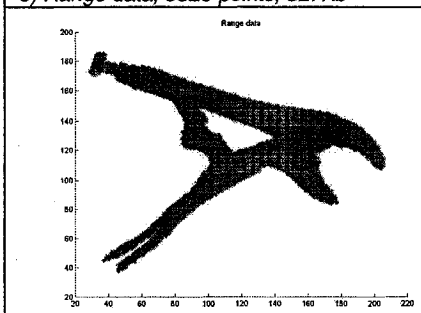
b) Range data, 5328 points, 327Kb | *Resulting model, 1.2 hrs, 83Kb*



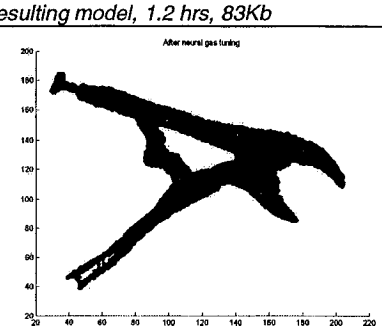
c) Range data, 5328 points, 327Kb



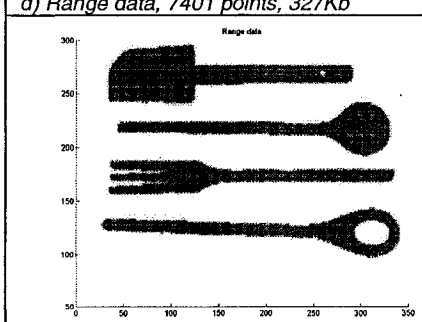
Resulting model, 1.2 hrs, 83Kb



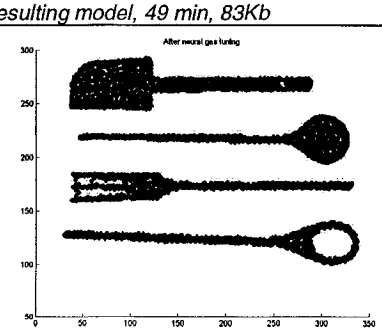
d) Range data, 7401 points, 327Kb



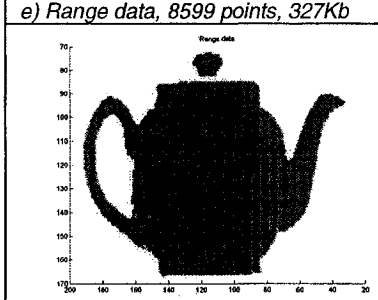
Resulting model, 49 min, 83Kb



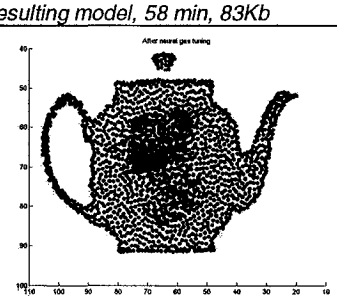
e) Range data, 8599 points, 327Kb



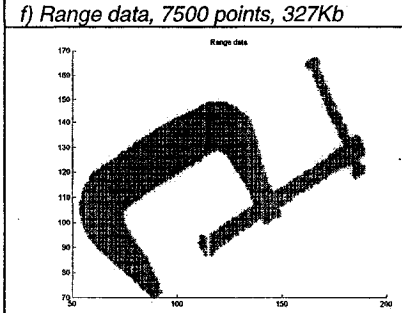
Resulting model, 58 min, 83Kb



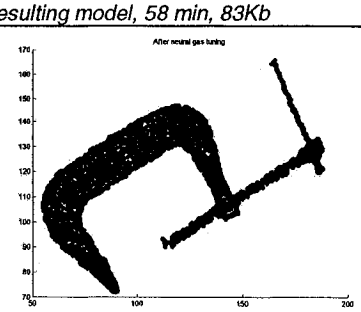
f) Range data, 7500 points, 327Kb



Resulting model, 58 min, 83Kb



g) Range data, 3834 points, 327Kb



Resulting model, 24 min, 83Kb

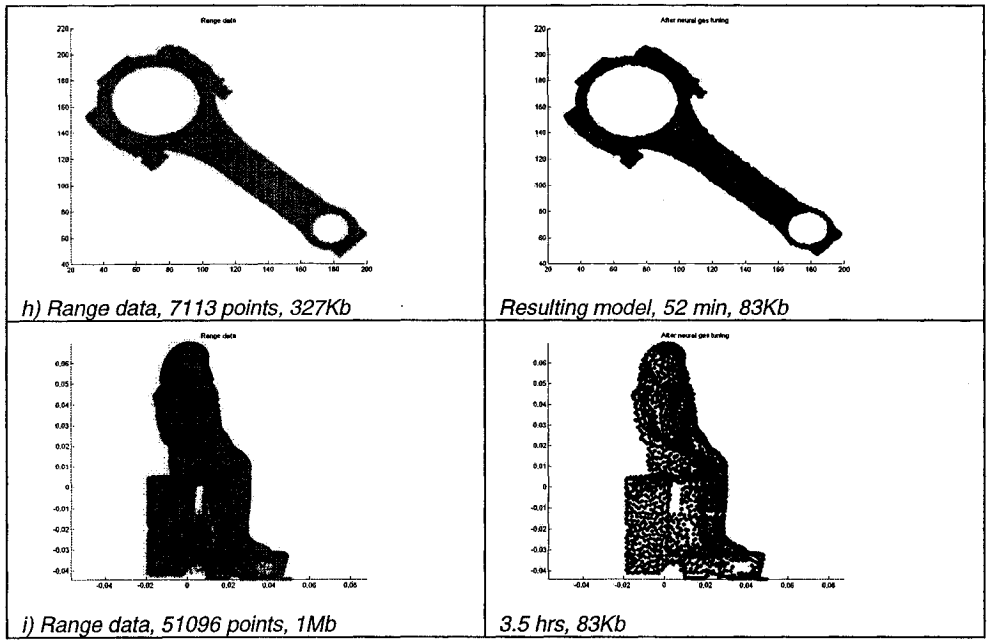
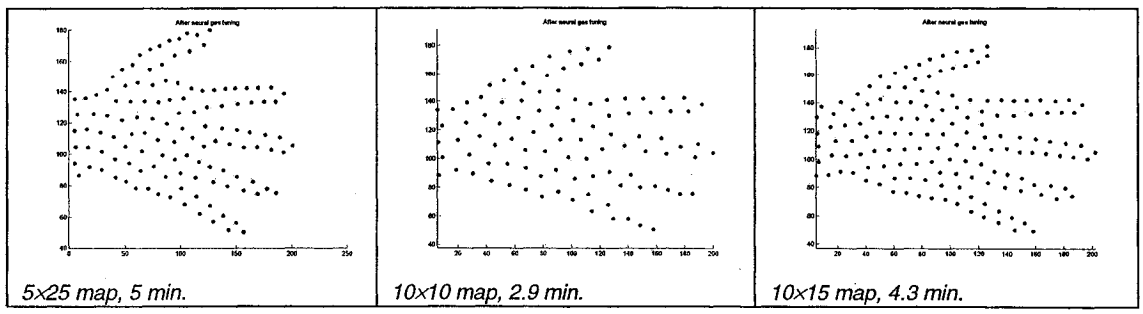


Figure 5.31. Neural gas models for a map of 45×55 and 20 training epochs

The neural gas network can model correctly objects with cavities (Fig.5.32b, e, h), small details (Fig.5.32b, d, g) and even entire scenes that contain many objects (Fig.5.32e). It has some problems for objects with thin edges or too close parts (Fig. 5.32c, d, e). It can also be remarked a waste of neurons where the data distribution is high (Fig. 5.32f, black areas on the pot).

The models that have to be stored are smaller than the original data set. One aspect that should be mentioned here is that the 327Kb size of the range data is not the real storage of the model. It is the size of the compressed model where only the z dimension is explicit, while the x and y are implicitly given by their position in the file. To store the real matrix it takes between 500Kb and 1.3Mb depending on the number of points that describe the object. The computation time of the neural gas network is quite long.

A number of simulations have been performed to show the effect of the map size in terms of computation time, storage and quality of the model. As expected, the quality improves with the map size and the number of training epochs (Fig.5.32, 5.33).



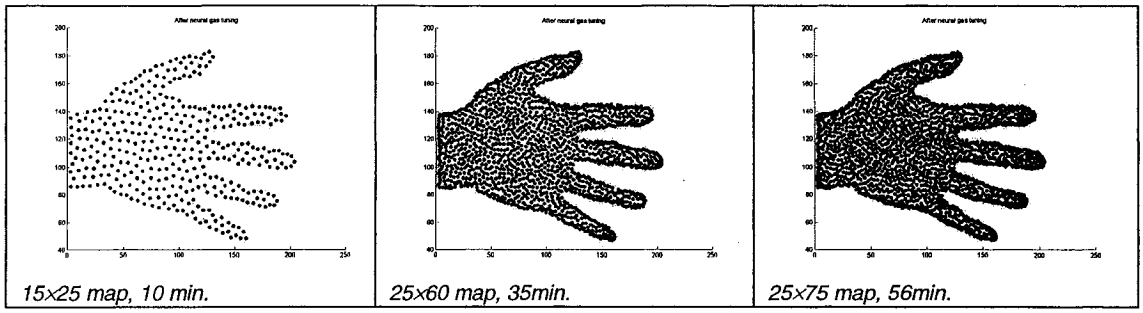


Figure 5.32. Influence of map size for 10 training epochs (19280 points)

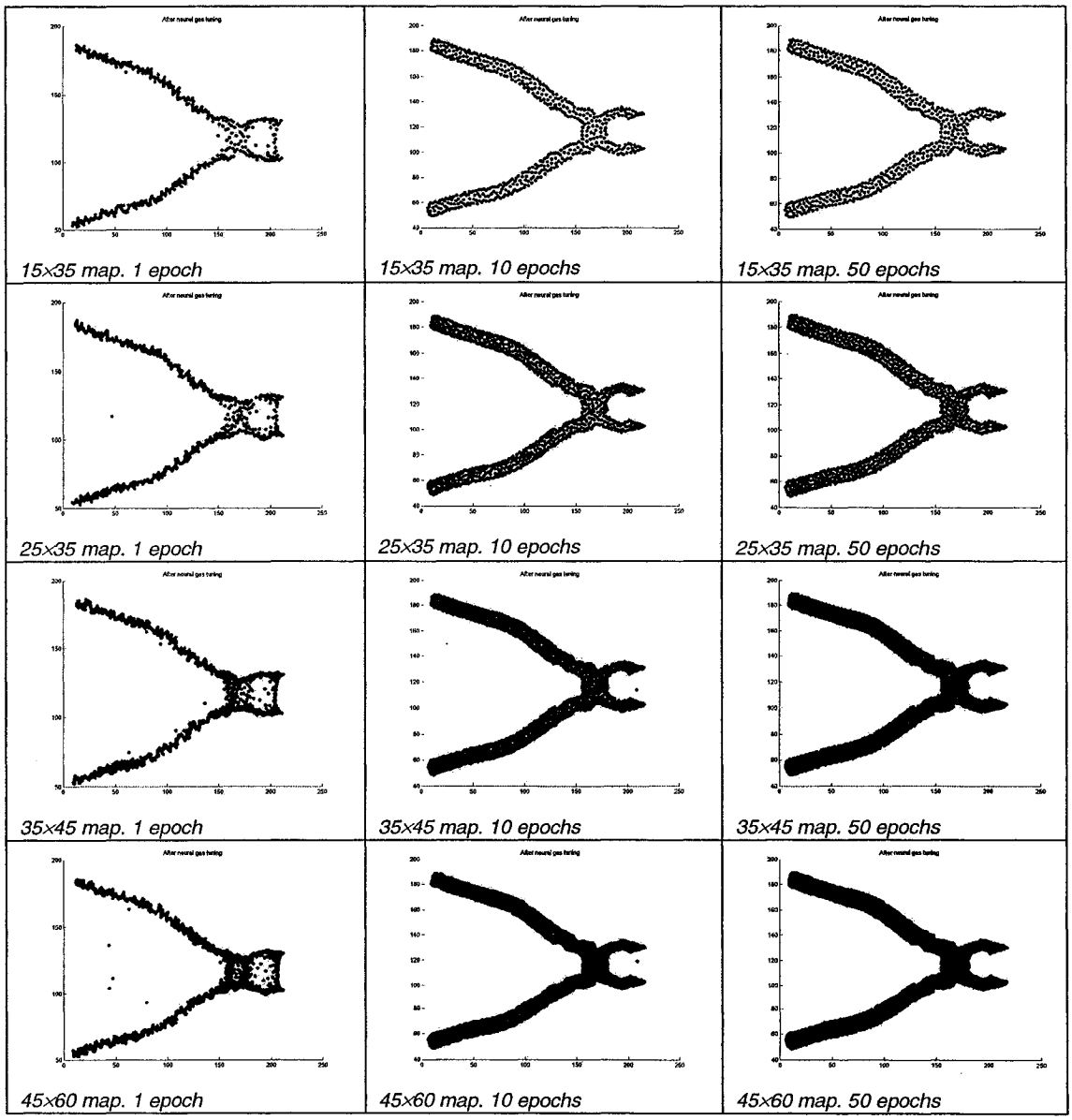


Figure 5.33. Influence of map size and training epochs

The same conclusion can be drawn from Fig. 5.34, which shows the evolution of the quantization error with the map size.

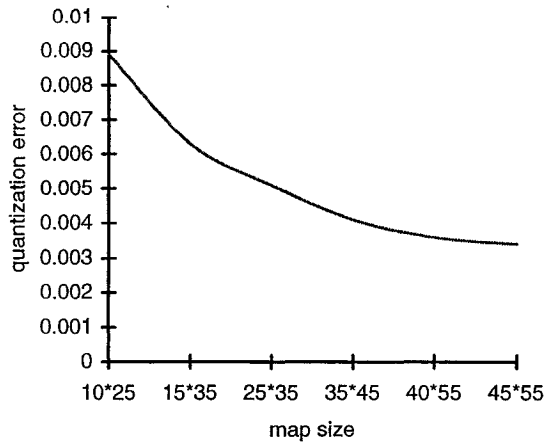


Figure 5.34. Evolution of quantization error as a function of the map size for a learning rate α_0 =0.5 and 10 training epochs

The quality also improves with the number of training epochs, but both these advantages come at the price of higher computational time (Fig. 5.32). Thus a compromise must be made between the desired quality of the model and the time a user is willing to pay to have the model built. An important consideration to be made here is the purpose of the model – if the single requirement is "image making", a medium map would suffice, with a technique to fill the empty space between neurons, such as point-based rendering. If the object is designed for simulations, a larger map and a larger number of training epochs should be used. For example, a map of size 15×25 (Fig. 5.32), would be enough for image making. A technique like point-based rendering [19] could improve the point-like aspect of the image (Fig. 5.35).

Figure 5.35. Point-based rendering of the hand

Point-based rendering can be considered as surface reconstruction from its point samples. The rendering of the reconstructed object surface is done in a meshless manner. A hierarchy of spheres of different radii is used to model a high-resolution model. For every surface point a sphere is associated with it such that neighbouring spheres are just overlapping to assure hole-free reconstruction. Together with each spherical sample, information on normals or material properties can be kept.

Table 5.7 shows the influence of the map size on the required storage space. It can be easily noticed that even for large map size the storage remains below 100Kb.

Table 5.7. Required storage according to map size

Map size	*Required storage space*
15×55	16Kb
15×35	20Kb
25×75	64Kb
30×50	52Kb
40×55	74Kb
45×55	83Kb
45×60	90Kb

Fig. 5.36 shows how the learning rate affects the quality of the resulting model for different numbers of training epochs and for a fixed map size of 35×55.

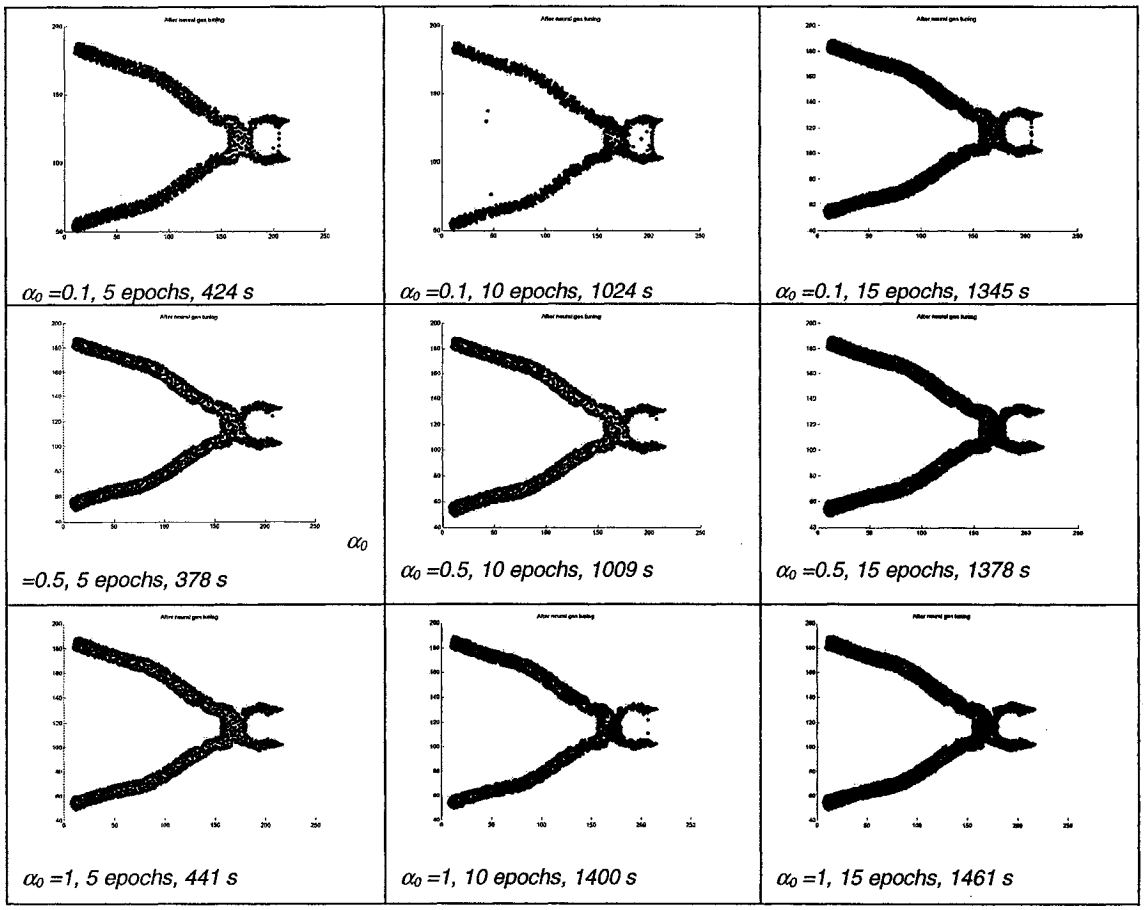


Figure 5.36. Influence of the initial learning rate α_0 for a fixed map size 35×55

It can be seen that the quality is improving with a larger value of the initial learning rate α_0 (typically the value of α_0 is in the interval [0,1]). This fact is also proven by Fig.5.37 and Fig.5.38, which show the evolution of the quantization error with α_0.

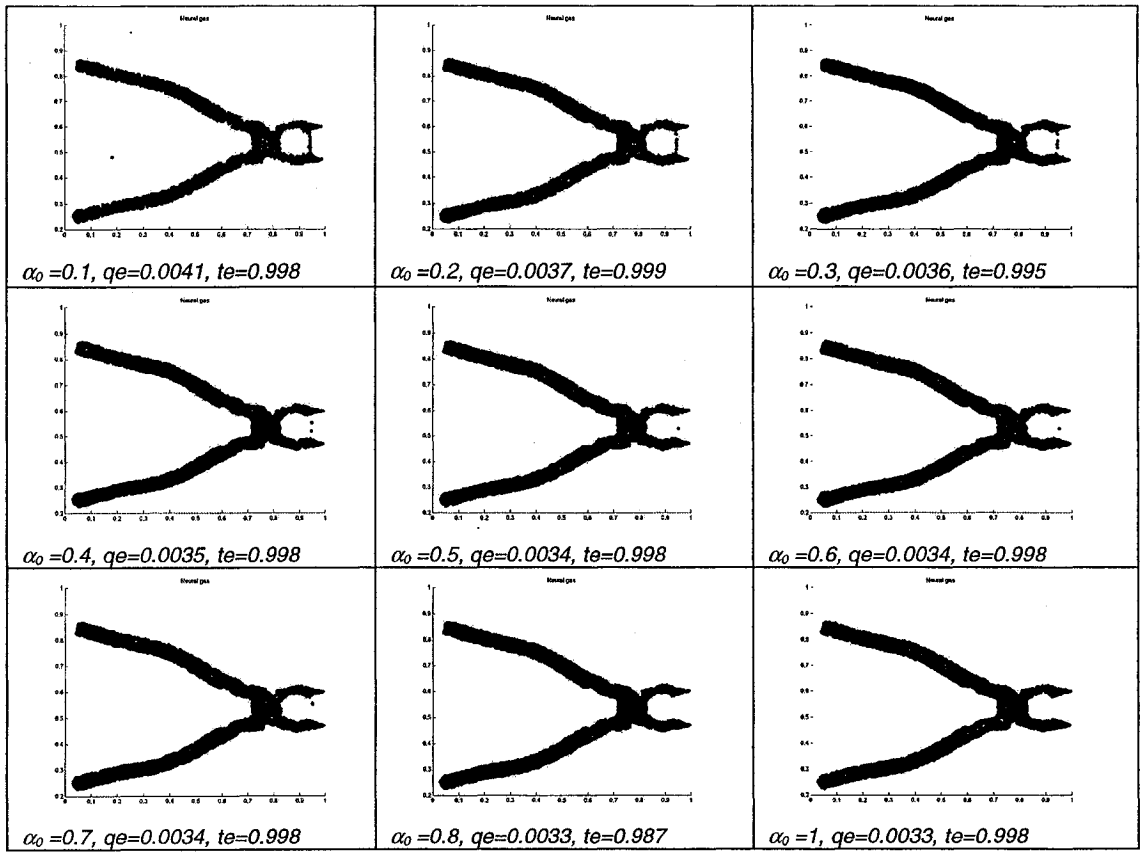


Figure 5.37. Error evolution with α_0 for a fixed map size 45×55 and 10 training epochs

The quantization error decreases slowly with an increase in the value of the learning rate, but remains approximately constant for a threshold larger or equal to $\alpha_0 = 0.5$.

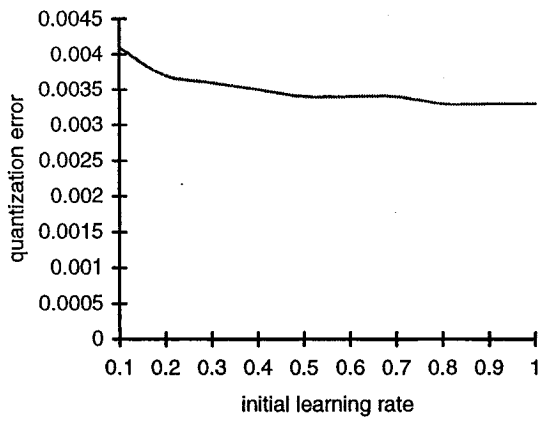


Figure 5.38. Evolution of quantization error with α_0

This value is actually used for all the models. It can also be seen that the time required for the same number of epochs is the same, so it does not depend on the value of the learning rate.

The last category of simulations is meant to elicit the influence of the initial value of the decay parameter λ_0 (eq. 4.18, section 4.3.3.3) over the learning procedure. For initial low values of the decay parameter, the network reveals to be not able to learn the shape of the object, as depicted in Fig. 5.39.

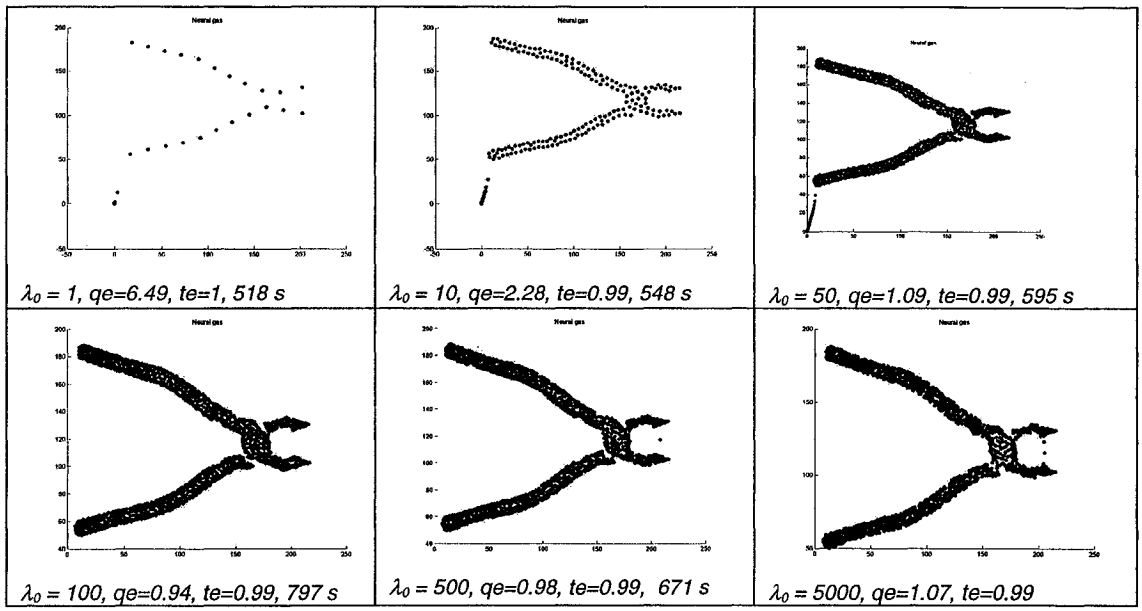


Figure 5.39. Influence of λ_0 for a fixed map size 25×45 and 10 training epochs

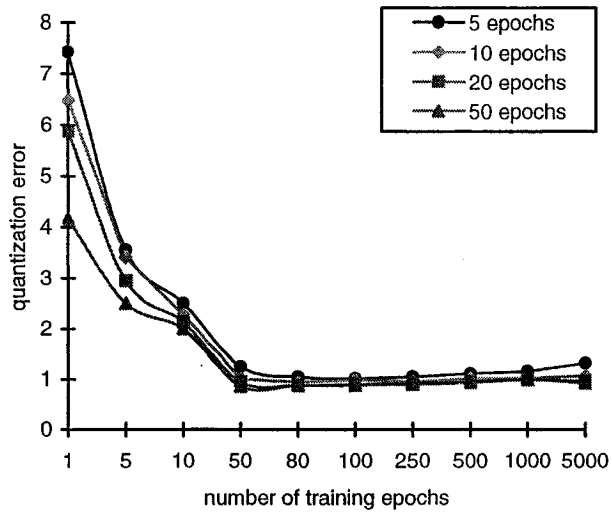


Figure 5.40. Influence of λ_0 on the quantization error for different number of training epochs

Fig. 5.40 shows a decrease in the value of errors with an increase in the value of λ_0 until the value of λ_0 = 80-100. After that point the error starts to increase slightly, and thus the quality deteriorates again. A value of approximately the number of neurons divided by 2 is used for the modeled objects.

Fig. 5.41 and 5.42 show the decrease in quantization error for the case when λ_0 =100. The more training epochs we use, the better it is, because nodes can be redistributed over the data space more efficiently.

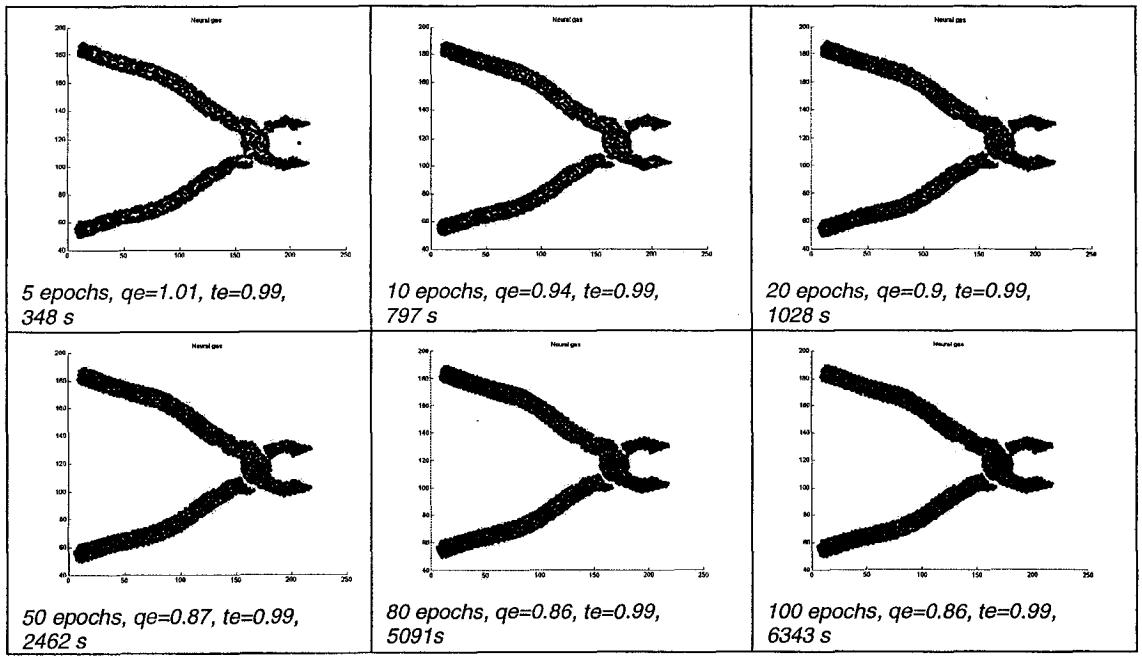


Figure 5.41. Evolution of error with training epochs for λ_0 = 100 and a fixed map size 25×45

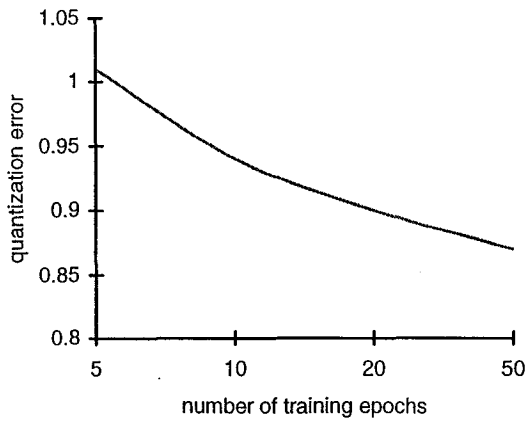


Figure 5.42. Evolution of quantization error with the number of epochs for λ_0=100

From all the above-mentioned aspects, it can be concluded that the neural gas provides a relatively low error level over many training samples, it is resilient to initial starting positions and variations in parameters' settings. This is gained at the expense of an increased computational load. The network is able to produce good results, even when given a limited time to converge.

5.2.3. Applications

5.2.3.1. Object Morphing

Based on the same principle discussed with the MLFF neural network, object morphs can be created by a linear interpolation of neural gas representation. In this case the matrix of weights is actually the representation of the model. Therefore, the interpolation is done between the two representations. Fig. 5.43 and 5.44 show the morphing of a pair of pliers in another pair of pliers and that of a doll face into a human face. In the latter example, the face is represented both from the front and the profile for clarity.

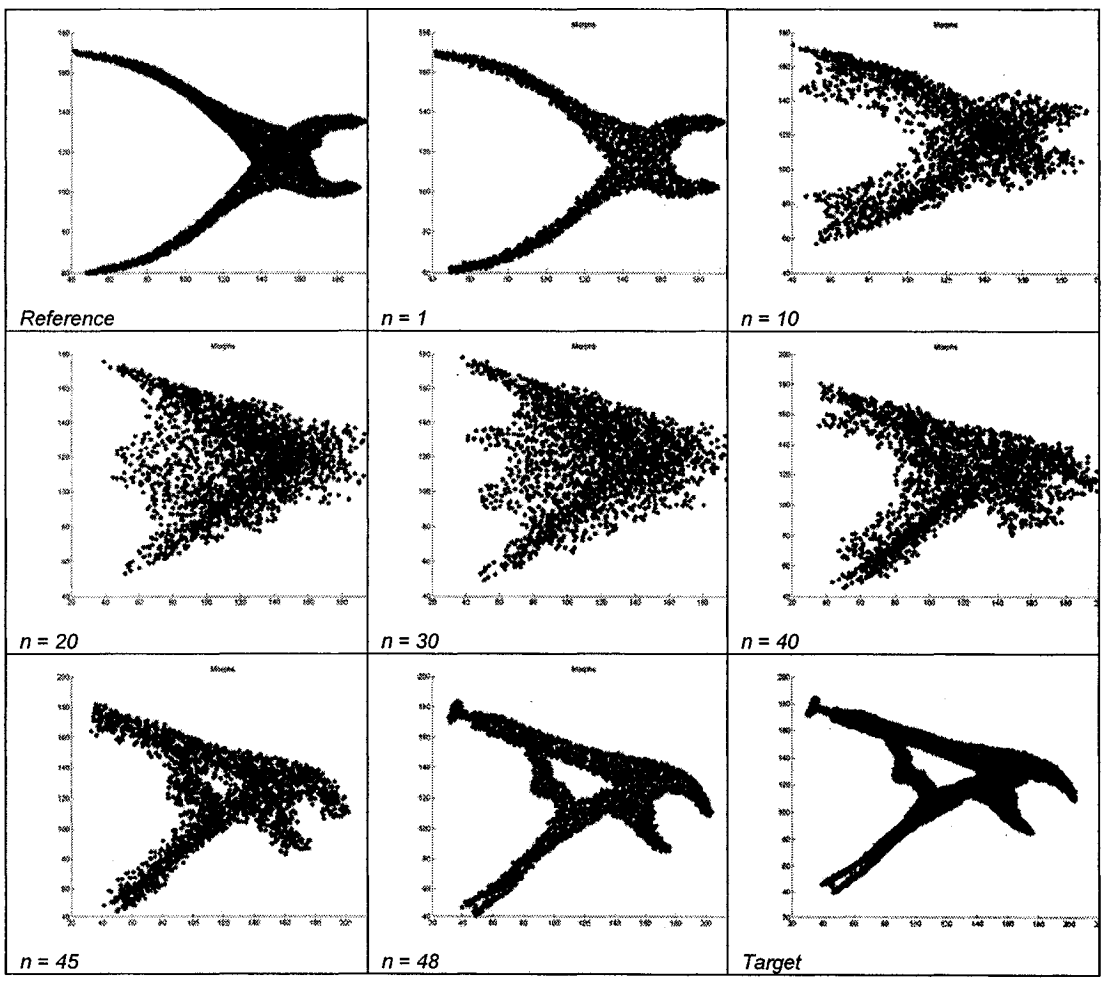
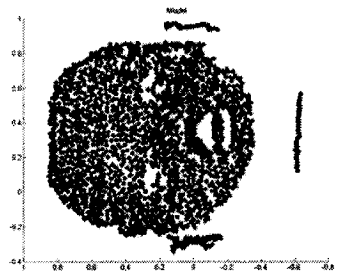
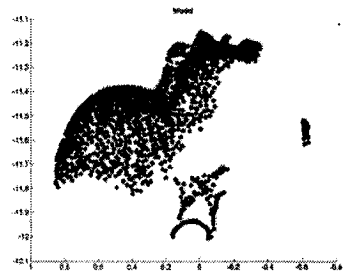
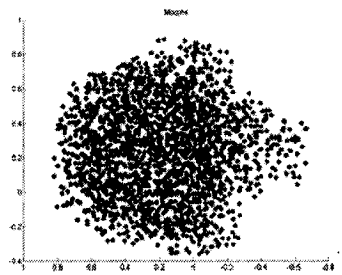
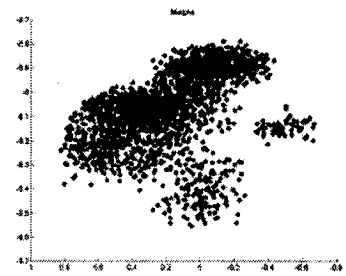
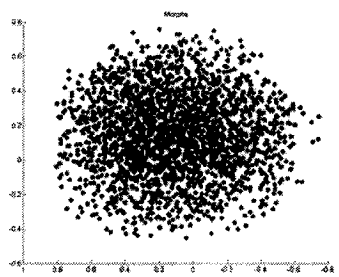
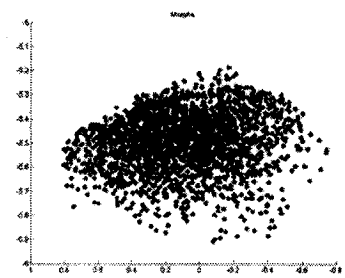
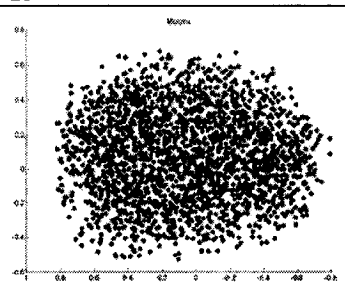
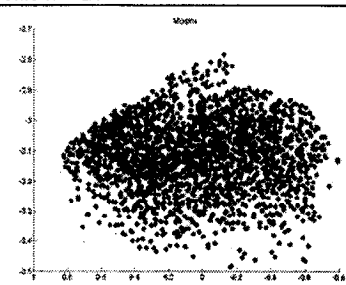
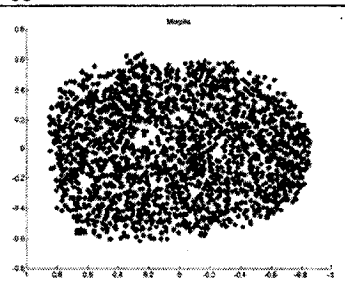
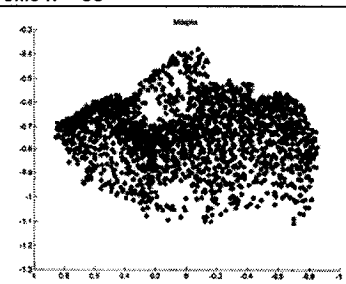


Figure 5.43. Morphing a pair of pliers in 50 steps

	
Reference model	*Profile reference model*
	
n = 10	*Profile n = 10*
	
n = 25	*Profile n = 25*
	
n = 35	*Profile n = 35*
	
n = 45	*Profile n = 45*

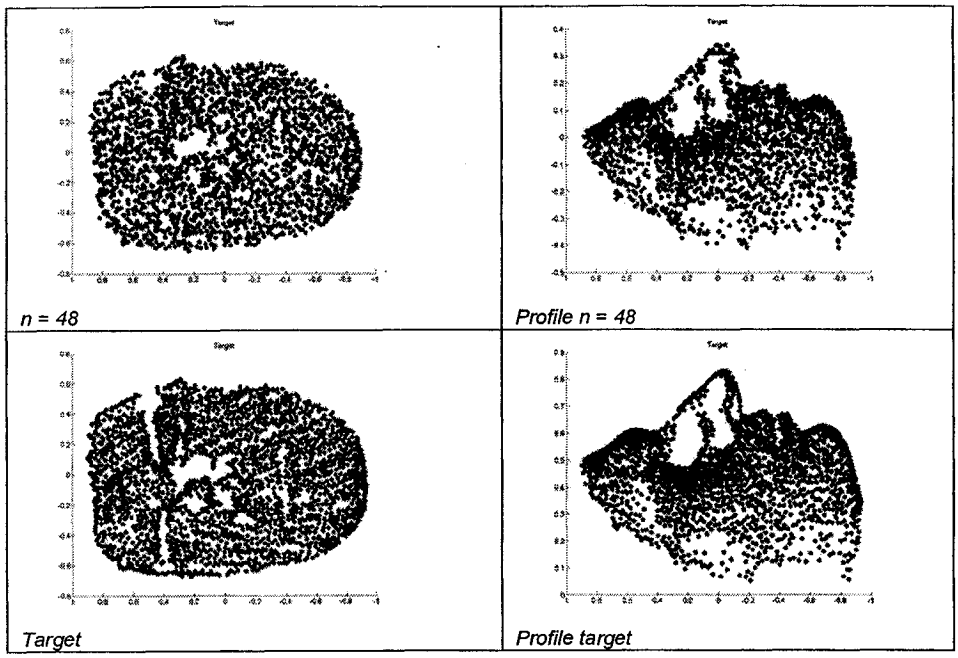


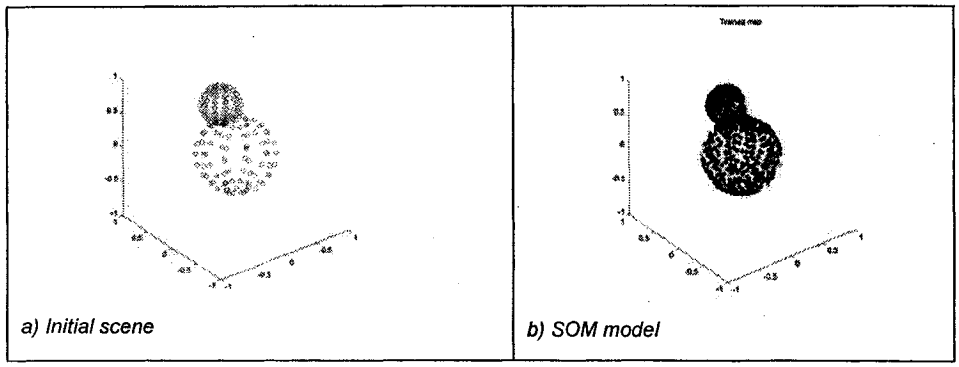
Figure 5.44. Morphing a doll face into a human face

5.2.3.2. Object Motion Estimation

The object motion estimation is performed in the same manner as in the case of MLFF neural network, using the transformation module. Nothing changes, neither in the procedure, nor in the results.

5.2.3.3. Segmentation

As both the SOM and the neural gas network are in fact vector quantization methods, they are able to preserve the structure of the input space and at the same time reduce the data set. This property makes them useful as a preprocessing technique on the data representing complex scenes, prior to clustering.



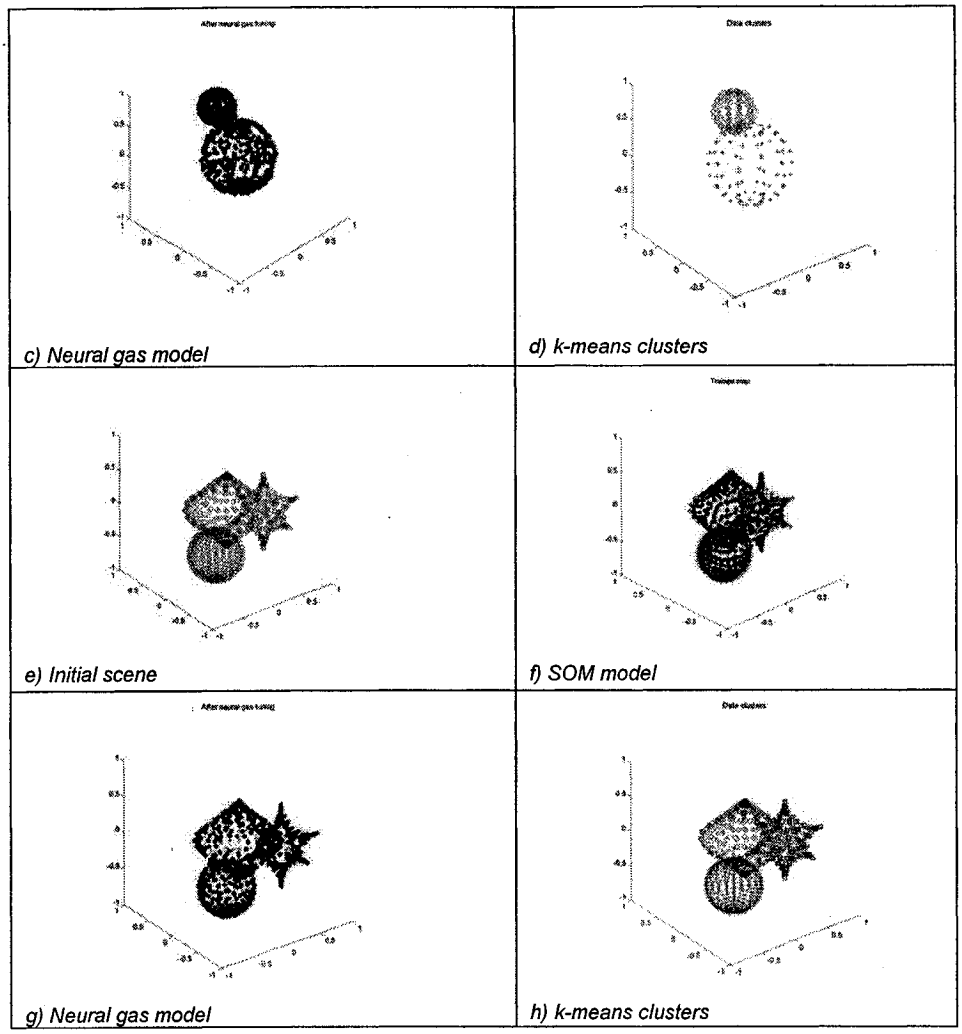


Figure 5.45. Segmentation using k-means clustering, SOM and neural gas

Tests are run for two scenarios (Fig. 5.45a, e) with just k-means clustering and then with both neural architectures prior to the use of k-means clustering to elicit the improvements given by them in the clustering procedure. The first scenario contains synthetic data of two barely touching spheres. The second one contains three objects situated very close to each other. The results are presented in Table 5.8. In both cases, the k-means clustering preceded by neural gas or SOM gives better results than in the case in which only k-means is applied alone.

Table 5.8. Segmentation results

Scene	*Misclassified points (%)*		
	k-means	*k-means + neural gas*	*k-means + SOM*
Scene 1	1	0.56	0.55
Scene 2	3.5	1.7	2.3

For the first scene, the result of SOM and k-means clustering are the same as the ones of neural gas with k-means. For the second test scene, the neural gas results are better, due to the boundary problem exhibited by SOM in modeling the star object (Fig. 5.45f).

Some more clustering results using range data and the neural gas network prior to k-means clustering are presented in Fig. 5.46.

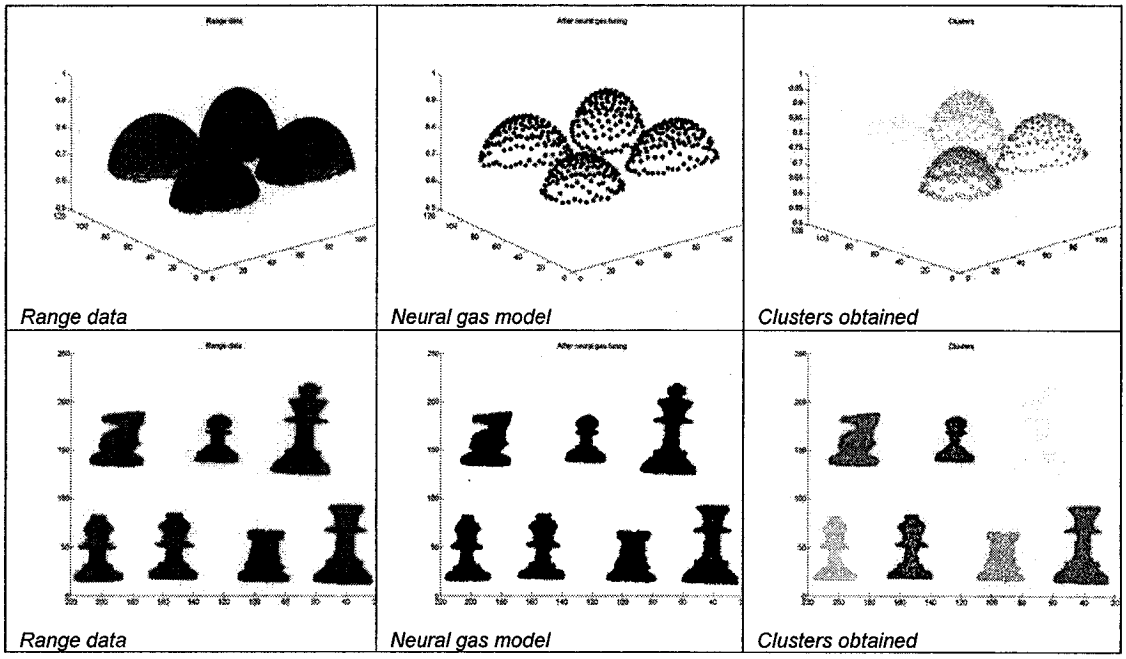


Figure 5.46. Clustering range data with neural gas and k-means

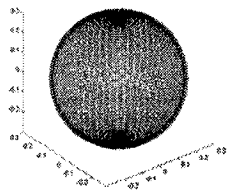
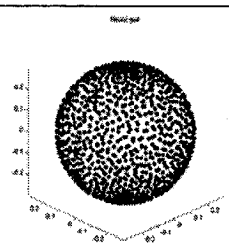
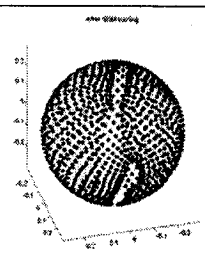
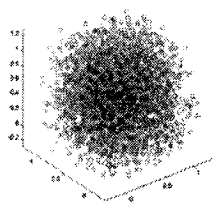
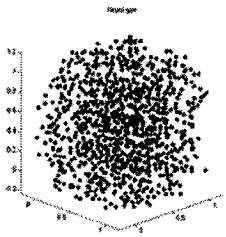
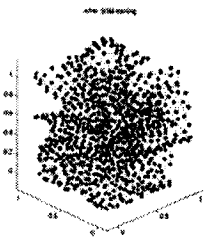
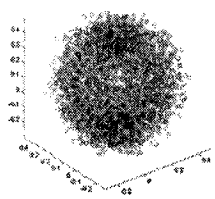
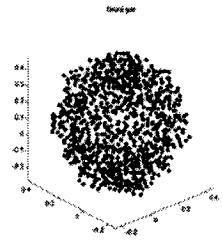
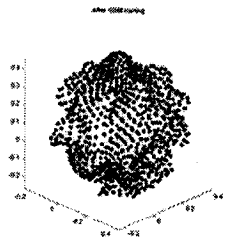
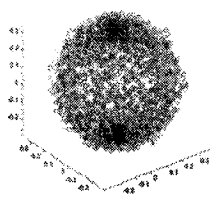
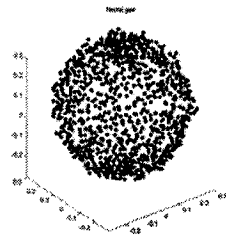
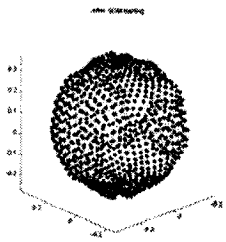
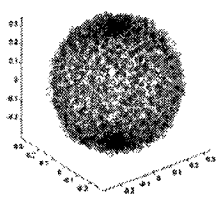
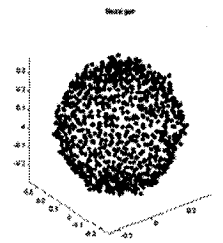
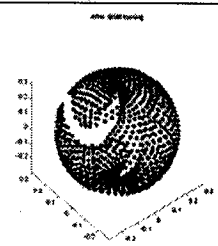
5.2.4. Self – Organizing Map - Neural Gas Comparison

Further tests are targeted to reveal differences and common points in the behaviour of the SOM and neural gas networks. The comparison is done here separately, because the two models, as it has been previously said, have both roots in the same neural self-organized architecture and their quality can be estimated in the same terms.

Data Sets

The same data sets have been used to build models for the two neural architectures. However, their behaviour in the presence of noisy training data is different.

Fig. 5.47 presents the results of training a neural gas network with a map size of 25×45 for 20 epochs, for α_0 =0.5, λ_0 = number of neurons/2 and a SOM, with the initial neighbourhood radius σ_0 = 5, and a map size of 25×45, trained for 100 epochs, with data corrupted with different levels of noise.

		
Training data set, 3721 points	*Neural Gas qe=0.0112, te=0.995*	*SOM qe=0.0133, te=0.0172*
		
Noisy data set, random 0 - 1	*Neural Gas qe=0.1388, te=0.98*	*SOM qe=0.0801, te=0.0339*
		
Noisy data set, random 0 – 0.2	*Neural Gas qe=0.0538, te=0.99*	*SOM qe=0.0304, te=0.0384*
		
Noisy data set, random 0 – 0.1	*Neural Gas qe=0.0383, te=0.99*	*SOM qe=0.0266, te=0.0156*
		
Noisy data set, random 0 – 0.05	*Neural Gas qe=0.224, te=0.99*	*SOM qe=0.0241, te=0.0161*

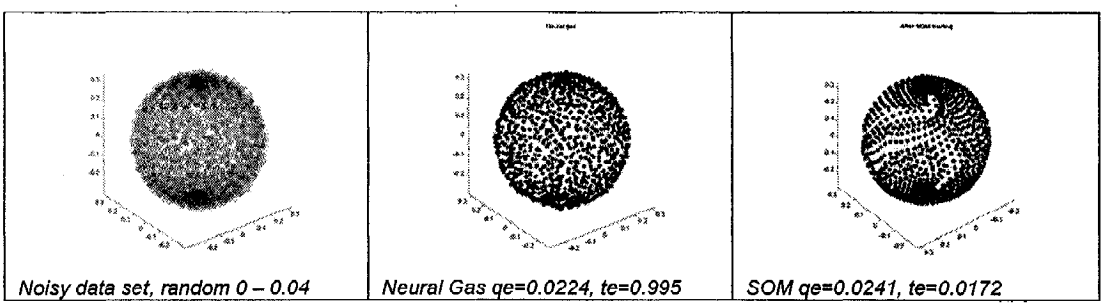


Figure 5.47. Influence of noisy data over the training procedure

It takes approximately 250s for the SOM to build a model, while it takes approximately double for the neural gas.

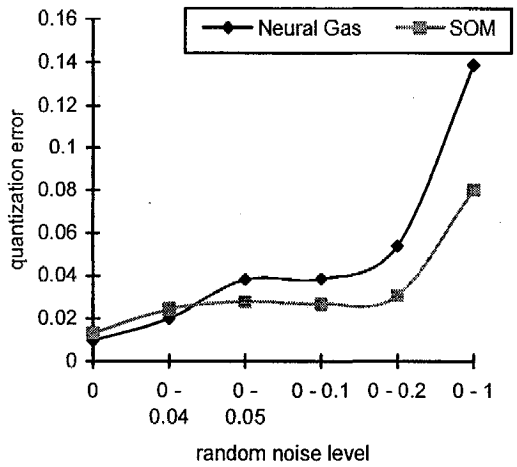


Figure 5.48. Evolution of quantization error in presence of random noise

The first thing that is noticed by visually comparing the results is that the SOM suffers from the boundary problem. The models obtained look as if they contain cavities. By comparing the quantization error in Fig. 5.47 and Fig. 5.48 it can be seen that for low levels of noise the neural gas performs better than the SOM (achieves lower levels of quantization error). For higher level of noise, the SOM tends to smooth the effect of noise, while the neural gas, having a very good accuracy, follows the noisy patterns.

A comparison of the topographic error as seen in Fig. 5.49 leads to the conclusion that while the SOM preserves topology, neural gas does not, as shown by the large value of the topographic error. However, for the SOM the level of topographic error increases with the level of noise, while for the neural gas it remains relatively constant.

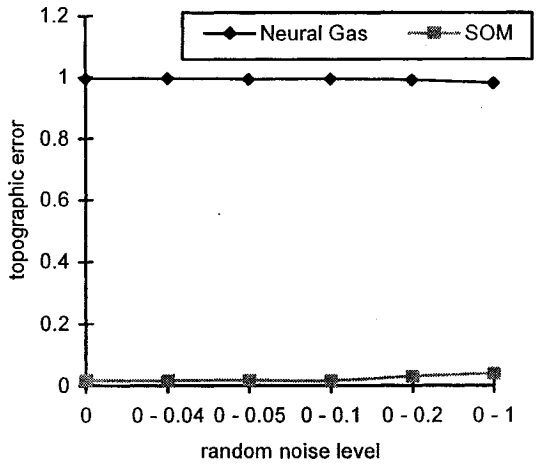
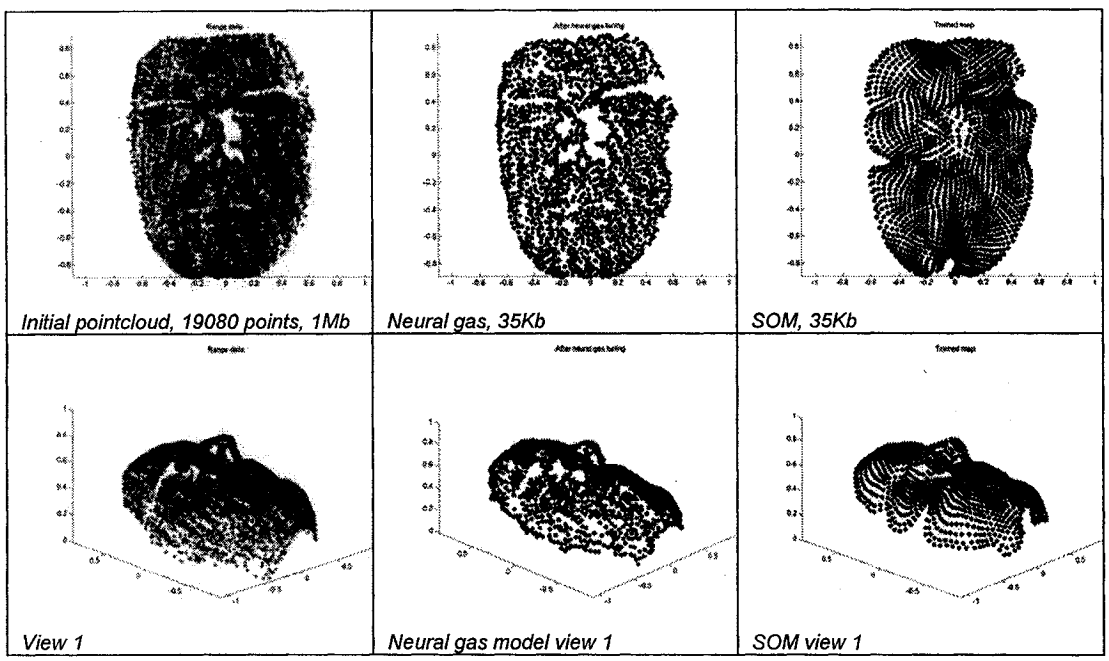


Figure 5.49. Evolution of topographic error in presence of random noise

A comparison in terms of modeling power with non-noisy data is presented in Fig.5.50 and Fig. 5.51. For 19080 points of the face in Fig. 5.50, it takes 26 min. to build a SOM model and 42 min. to build the neural gas model for the same map size. The time is almost double for the neural gas model. The resulting models have almost the same storage space requirements. The SOM model suffers from the boundary problem and tends to smooth details, while the neural gas network model follows small sharp details in the modeled object. This is due to the fact that the neurons are released from the fixed grid of SOM. The same property makes it easy for the neural gas to model an entire scene of objects and makes it impossible for the SOM model.



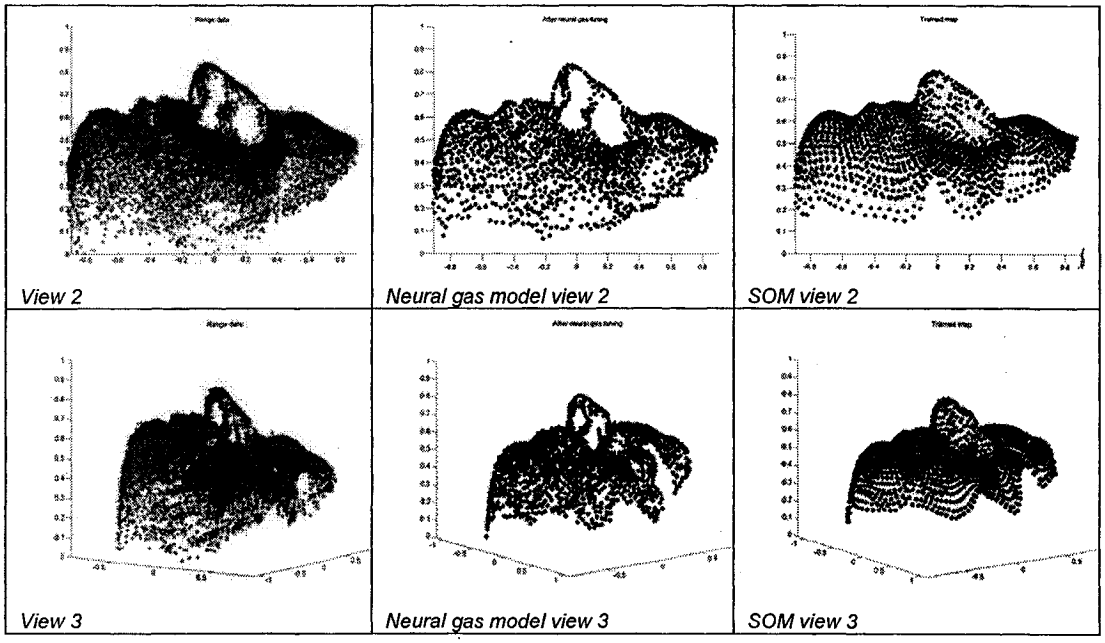
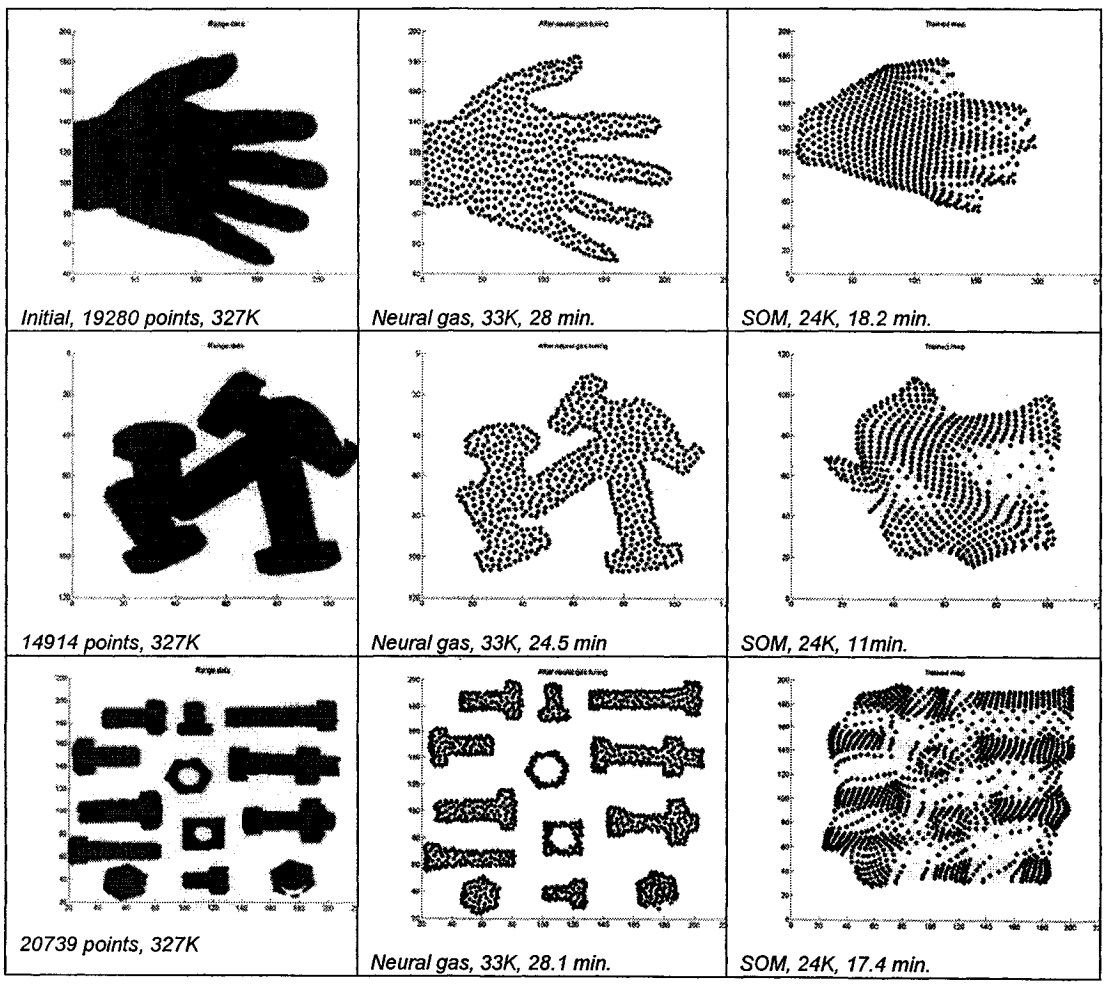


Figure 5.50. 3D comparison between Neural Gas and SOM models



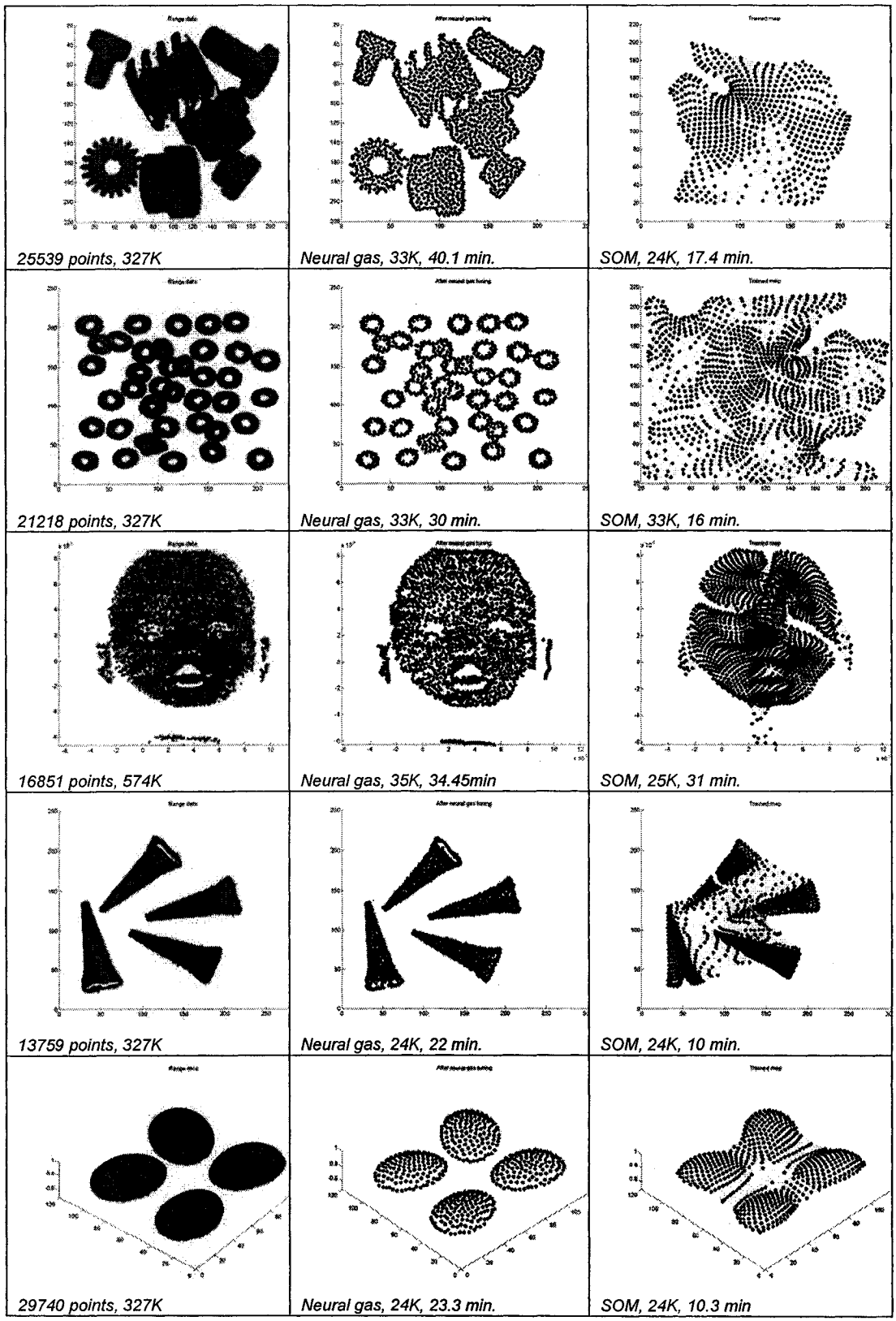


Fig. 5.51. Models for a fixed map size of 25×35

As visually seen in the figures above and as depicted in Fig. 5.52, the quality of the neural gas model appears to be better. In both cases, the quality is improving with the number of training epochs.

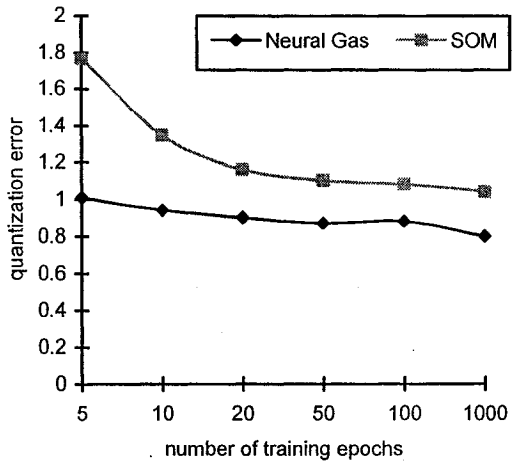


Fig. 5.52. Evolution of quantization error with the number of training epochs

Neural gas does not preserve the topology. The evolution of the topographic error with the number of training epochs is presented in Fig. 5.53. For neural gas, there is no change in the topology preservation. For the SOM, the error decreases in the beginning with the number of training epochs, remains constant for a while, and then rises again.

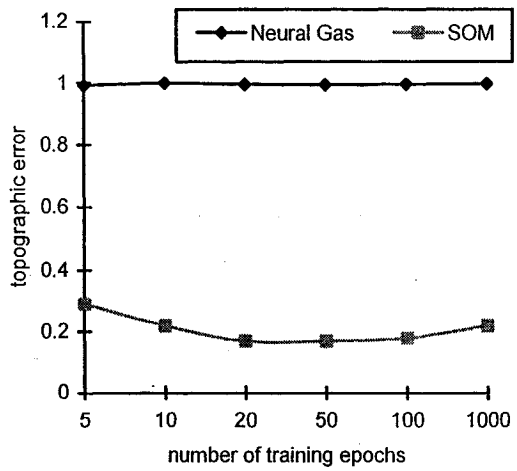


Fig. 5.53. Evolution of topographic error with the number of training epochs

In terms of applications, there are the same uses for both models. However, because of the boundary problem, the SOM models are to be avoided for non-noisy data.

5.3. A Comparison Between Multilayer Feedforward and Self-Organizing Models

5.3.1. Methods to Define the Objects

From the point of view of the methods used to define the objects, all resulting models are defined by a matrix of weights. The initial pointcloud of range data or synthetic data is first normalized and is sequentially supplied to the inputs of the neural architecture. After the learning procedure, the weights of the neural network embed the representation of the object.

This representation has to be accompanied by the network architecture (2 digits, representing the number of neurons in each hidden layer) to allow for the reconstruction of an object defined by a MLFF neural network. For the case of SOM and neural gas, the object representation is inherent in the matrix of weights, which in this case contains the coordinates of points in space where the neurons that represent the objects are located.

5.3.2. Purpose of the Object Model

Regarding the purpose of the object models, there are two main reasons for building 3D objects: "image making" and simulation [47]. While "image making" requires that the model looks good (is realistic, sufficiently complex and conforms to the geometry of the model), simulation requires accurate models. While the first one is described as interactive and approximate, the other one is generally slow but precise. The MLFF models can be used both for "image-making", since they look good, are sufficiently complex and conform to the geometry of the modeled object. They also reveal to be appropriate for simulation since they can be accurate if the user is willing to pay the computational cost. SOM models are more suitable for simulation, as they present a good topology preservation and are accurate even in the presence of noise, but are not suitable for "image-making" since they suffer from the boundary problem. The neural gas models are again more suitable to simulation since they are very accurate (as long as the training data is not too noisy), but a supplementary technique like point-based rendering is needed to make them look better and avoid the point-like aspect of the modeled objects.

5.3.3. Computational Cost

With regards to computational cost matters, there are three aspects to be considered: the construction (training) time, the display cost (generation time) and the storage space.

First, considering the training time, the cheapest in terms of computational time is the SOM followed by the neural gas network. The MLFF neural network is the most expensive. This is the reason why only a couple of complex models are built using this network, as depicted in Fig. 5.54 and section 5.1.

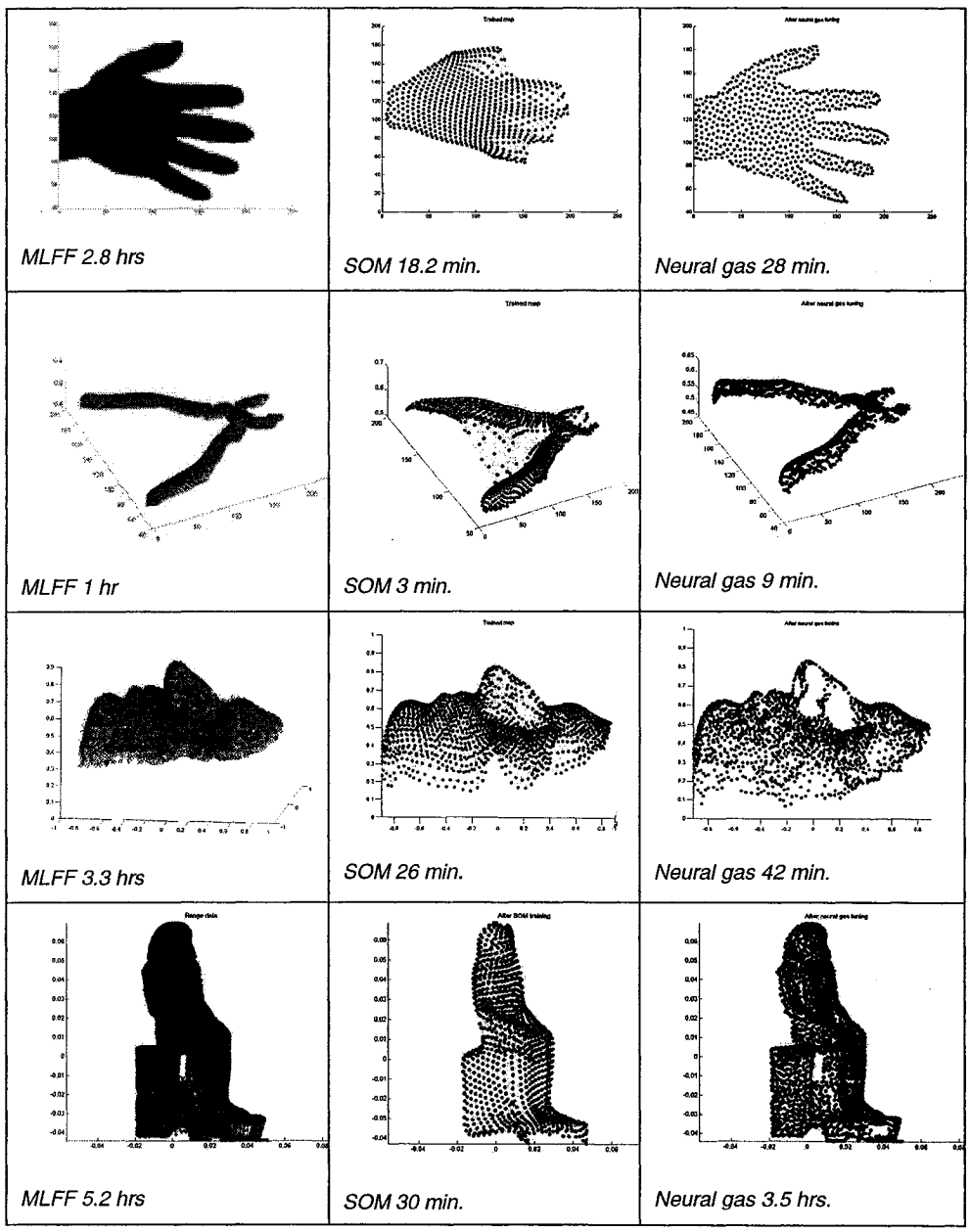


Figure 5.54. MLFF, SOM, neural gas comparison in training time and visual quality

In addition, besides the training time, the MLFF network needs also a long generation time (up to 10 minutes), unlike the two self-organized architectures. The network has to be evaluated for points in the [–1 1 –1 1 –1 1] range and all the points inside the object displayed,

while for the other two architectures, the generation time is equivalent to the display time. Actually none of the models provides a real-time behaviour. This is mainly due to the platform used to perform the simulation. It is known that Matlab is not efficient in terms of computational time, but in this stage, the sole purpose was to show that it is possible to model objects using neural networks and to elicit some properties of these representations. A serious enhancement is expected by migrating to a different platform such as C, or by using a hardware implementation of the neural network architecture.

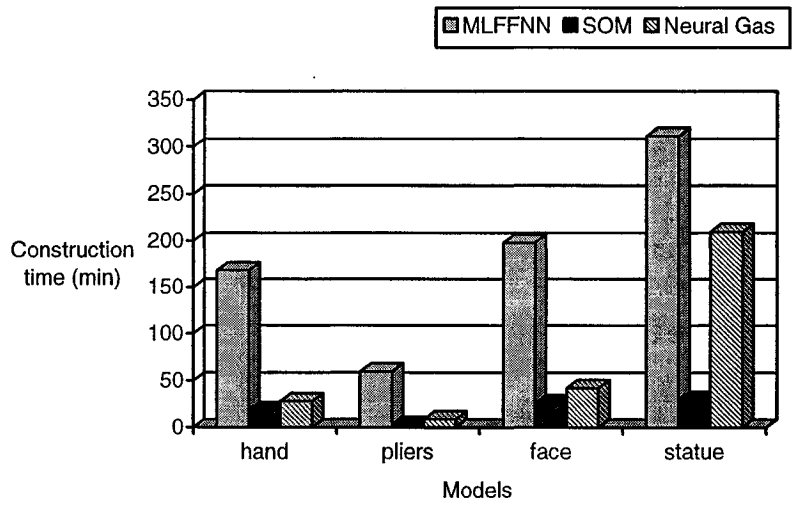


Figure 5.55. Comparison of construction time

As for the storage space aspect, all three models reduce significantly the space needed to store the representation of the model in comparison with the initial pointcloud.

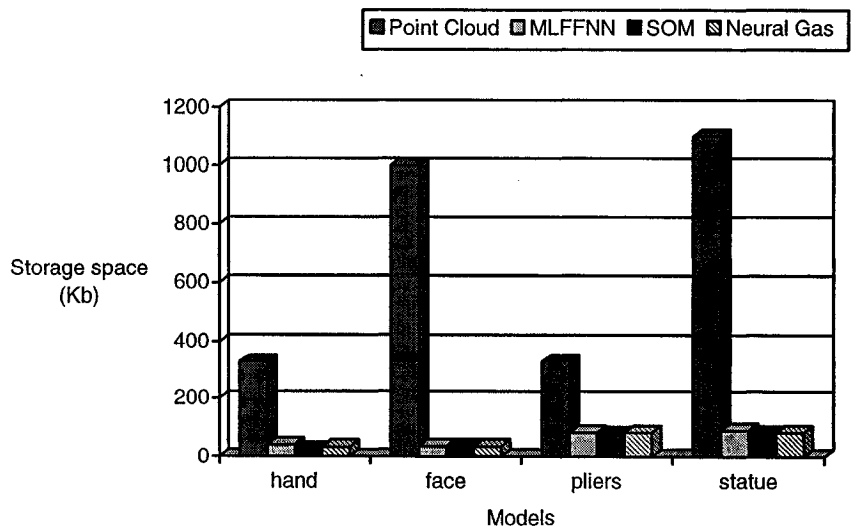


Figure 5.56. Comparison of required storage space

For some models, the space required to store the MLFF representation is a bit larger than the one required by the self-organizing architectures, but generally there is no significant difference between the compared neural architectures.

5.3.4. Complexity

The neural representations provided by the proposed neural architectures are simple and compact. They can be seen as a matrix or string that contains the values of the weights and the neural architecture in case of the MLFF neural network.

5.3.5. Conformance and Convenience

Conformance measures the degree to which the model actually represents the desired objects and provides a realistic, geometrically correct representation for the modeled object. Therefore, from the conformance viewpoint, the MLFF neural network deserves the first position. This is due to its volumetric properties and to its property of theoretically infinite resolution (only limited by the number of points used to test the model). Although it does not have the same accuracy as the neural gas network, the objects do not present the point-like aspects. However, when supplemented by point-based rendering, the neural gas takes the leading position. On the other hand, neither the neural gas, nor the SOM models offer the property of infinite resolution. In spite of its topological preservation capability, the SOM gets the last position. It suffers from the boundary problem, and the modeled objects do not look good.

Regarding convenience, neural gas and SOM do not permit extensive study on the modeled objects. They built a sparse model of an object and do not offer means to test if a point is inside or outside the model. In contrast, the MLFF representation is continuous and due to the volumetric and implicit properties, a MLFF network permits an extensive study of not only the modeled object, but of the entire object space.

5.3.6. Ease of Model Manipulation

Cascaded with the transformation module, all models can be easily transformed in size, position and shape (affine transformation, deformations and morphing can be easily performed regardless of the representation). However, the capability of building new objects by means of set operations is restricted to the MLFF neural network.

5.3.7. Applications

The inherent volumetric and implicit representation provided by the MLFF neural network makes it easy to be used for modeling morphs, set operations and for detecting object collisions in virtualized reality scenes. Cascaded with the transformation module it can also be used for object recognition or object motion estimation. The self-organized architectures do not pose volumetric properties, thus their use is restricted to the modeling of morphs. Moreover, the representation is not implicit, and therefore there is no way to test if a point belongs or not of an object, offering no means for detecting a match between two objects. This implies that they cannot be used for recognition either. They can be used however for motion estimation and also for object segmentation in complex scenes.

All things considered, it can be concluded that, because of the boundary problem, the SOM models are to be avoided for non-noisy data. Depending on the application requirements, one should choose between the MLFF neural network and the neural gas network representation supplemented with a point-based rendering technique. While providing a quite accurate model, with theoretically infinite resolution, with facilities for providing means for performing volumetric and implicit operations, the MLFF representation implies a computationally expensive training phase and also some time investment for the generation of the modeled object. Neural gas offers very good accuracy and a reduced computation cost. When supplemented with the point-based rendering, it can offer good-looking models. However, its range of use in applications is reduced to segmentation, morphing and motion estimation.

Chapter 6

Conclusions and Future Work

This study demonstrates the relevance of using neural networks for 3D object modeling and puts in evidence their current limitations, opening the door for further developments to improve the performance and proposing some promising paths for solutions.

6.1. Summary

3D modeling is an important issue in the context of virtualized reality. This thesis has explored three neural architectures that can be used to build 3D representations of objects.

Given a pointcloud of either range data or synthetic data that embeds the shape of the object to be modeled, a volumetric representation has been built using a multilayered feedforward neural network, with sigmoid activation functions. The network replies with a negative value proportional to the distance to the modeled object's surface if the point given as input is located in the interior of the modeled object, with zero if the point is located on the modeled object's

surface or with a positive value proportional to the distance to the modeled object's surface if the point given as input is located in the exterior of the modeled object. The network can be operated in two modes: the generation mode, in which the user knows the equation of the modeled object and sets the weights such that the network implements the desired operation, or in a training mode, in which, provided with a pointcloud, the network learns a volumetric representation of the object embedded in the pointcloud. Chapter 3 presented the implementation issues, the parameters' adjustments and some possible applications for the MLFF neural network. Modeling results obtained with this architecture and their interpretations are contained in section 5.1. Our main observations are that the representation is simple, compact and provides a good accuracy. However, the computational workload is expensive. The representation does not offer real-time behaviour. On the other hand, it can offer a theoretically infinite resolution, it is continuous and permits an extensive study of not only the modeled object, but also of the entire object space. Thus it inherits the advantages of the polygonal and surface models, without inheriting their drawbacks. It can be used to perform simple operations such as object morphing, set operations and object collision detection.

The representation given by the two self-organized architectures has the characteristics of a surface model. The models are obtained faster and can achieve higher accuracy in less computation time than the multilayer feedforward counterpart. But unlike the multilayer feedforward network, they do not pose volumetric and implicit properties and therefore their use is restricted to object morphing. However, they can be used for object segmentation is complex scenes. Chapter 4 is dedicated to the implementation issues of the self-organizing architectures, their parameters' adjustments and some possible applications, while section 5.2, presents modeling results and their interpretations.

For both types of architectures, the representation module can be cascaded with a transformation module, whose role is to ensure an easy manipulation of the represented objects in size, position (modeling affine transformations) and shape (modeling deformations). Moreover, the module can be used to learn transformation/deformation information, when it is provided with a reference object and a transformed/deformed object. Cascaded with the transformation module the multilayer feedforward architecture can be used for object recognition. For all of the three architectures, the representation module cascaded with the transformation module can be used for motion estimation.

Section 5.3 presents a critical comparison of the three architectures in terms of purpose, computational cost, complexity, conformance, convenience, ease of manipulation and possible applications.

6.2. Contributions

As proposed in the beginning of this thesis, the main contributions are related to the analysis of neural network architectures for 3D object modeling in the context of virtualized reality. The main points that have been achieved through this work are:

- A comprehensive analysis and comparison of three neural architectures, one classical and two more recent, taking into account their modeling power, computational cost, complexity, conformance, convenience, ease of manipulation.
- A study of some possible applications of the neural architectures in the virtualized reality framework.
- The extension of a 2D object modeling architecture to the 3D case
- The introduction of a neural network deformation module that allows for generation and recuperation of deformation parameters of deformed 3D objects represented by neural networks.

A part of the obtained results in this work are presented in a research paper in Fall 2003 [102].

6.3. Future Work

Further developments of the work realized in this thesis will examine some more alternative neural architectures that could improve the learning time and ensure a real-time behaviour for at least the model generation.

In this context, migration from Matlab to a more highly-performance platform such as C, or a hardware implementation of the neural network architecture are seen as possible solutions for achieving this goal. Even though it possesses a high topology preservation and very good accuracy, the SOM network's modeling capability is limited in 3D object modeling due to its boundary problem. Thus a possible improvement could be the use of dedicated algorithms to treat the boundary distinctly from the entire object surface and thus avoid the holes created in the modeled objects' surfaces. Also, as briefly mentioned in chapter 5, a point-based rendering solution would help to improve the quality and performance of the models. Other rendering strategies could also be taken into consideration.

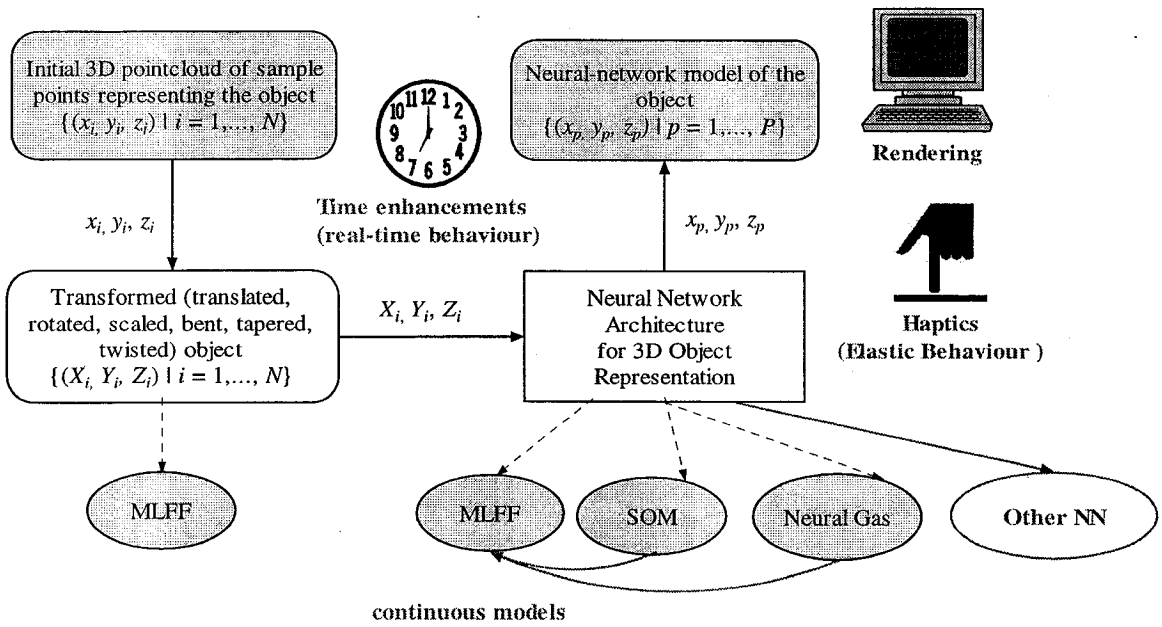


Figure 6.1. Future work

Finally, the object modeling approaches proposed in this thesis only account for the geometric shape of an object, not for its tactile properties. A future development is the addition of elastic (strain-stress) material properties to the modeled objects.

References

1. T. Kanade, P.J. Narayanan, and P.W. Rander, "Virtualized Reality: Concepts and Early Results", *Proceedings IEEE Workshop on Representation of Visual Scenes*, 1995, pp. 69–76.

2. T. Kanade, P. Rander, and P.J. Narayanan, "Virtualized Reality: Constructing Virtual Worlds from Real Scenes", *Proceedings IEEE Visualization*, 1997, pp. 277-283.

3. E.M. Petriu, "Neural Networks for Measurement and Instrumentation in Virtual Environments," *Neural Networks for Instrumentation, Measurement and Related Industrial Applications*, Eds. S. Ablameyko, L. Goras, M. Gori, V. Piuri, NATO Science Series, Series III: Computer and System Sciences vol. 185, IOS Press, 2003, pp. 273-290.

4. P. Boulanger, J. Taylor, S. El-Hakim, and M. Rioux, "How to Virtualize Reality: An Application to the Re-creation of World Heritage Sites", *VSMM98*, International Society on Virtual Systems and Multimedia, 1998, vol. I, 18-20, pp. 39 – 45.

5. M.G.-H. Mostafa, S.M. Yamany, and A.A. Farag, "Integrating Shape from Shading and Range Data Using Neural Networks", *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999, vol. 2, pp. 2015-2020.

6. A.W.K. Loh, M.C. Robey, and G.A. West, "Refining 3D Models Using a Two-Stage Neural Network-Based Iterative Process", *Proceedings 16th International Conference on Pattern Recognition*, 2002, vol.1, pp. 172 –175.

7. T. Lilienblum, P. Albrecht, and B. Michaelis, "Improvement of 3-D data By Neural Networks", *Proceedings International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing*, 1996, pp. 203 –211.

8. T. Lilienblum, P. Albrecht, and B. Michaelis, "3D-measurement of Geometrical Shapes By Photogrammetry and Neural Networks", *Proceedings 13th International Conference on Pattern Recognition*, 1996, vol. 3, pp. 330 –334.

9. C. Aldrich, "Visualization of Transformed Multivariate Data Sets with Autoassociative Neural Networks", *Pattern Recognition Letters*, 1998, vol. 19, issue 8, pp. 749-764.

10. S. Kaski, "Data Exploration Using Self-organizing Maps", *Mathematics, Computing and Management in Engineering* Series No. 82, Espoo 1997, Finnish Academy of Technology, [Online]. Available: *http://www.cis.hut.fi/~sami/thesis/node20.html*

11. M. Brown, and C.J. Harris, "Neural Networks for Modelling and Control", in *Advances in Intelligent Control*, Ed. C. J. Harris, Taylor&Francis, 1994, pp. 17-55.

12. J.-H. Hsu, and C.-S. Tseng, "Application of Three-dimensional Orthogonal Neural Network to Craniomaxillary Reconstruction", *Computerized Medical Imaging and Graphics*, 2001, vol.25, pp. 477-482.

13. G.K, Knopf, and J. Kofman, "Adaptive Reconstruction of Free-form Surfaces Using Bernstein Basis Function Networks", *Engineering Applications of Artificial Intelligence*, 2001, vol. 14, issue 5, pp. 577-588.

14. G.K. Knopf, and R. Al-Naji, "Adaptive Reconstruction of Bone Geometry from Serial Cross-Sections", *Artificial Intelligence in Engineering*, 2001, vol. 15, pp. 227-239.

15. J.-N. Hwang, and Y.-H. Tseng, "3D Motion Estimation Using Single Perspective Sparse Range Data via Surface Reconstruction Neural Networks", *IEEE International Conference on Neural Networks*, 1993, , vol.3, pp. 1696 –1701.

16. J.-N. Hwang, and H. Li, "A Translation/Rotation/Scaling/Occlusion Invariant Neural Network for 2D/3D Object Classification", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1992, vol. 2, pp. 397 –400.

17. J. Barhak, and A. Fischer, "Parameterization and Reconstruction from 3D Scattered Points Based on Neural Network and PDE Techniques", *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, issue 1, 2001, pp.1 –16.

18. J. Barhak, and A. Fischer, "Adaptive Reconstruction of Freeform Objects with 3D SOM Neural Network Grids", *Computers and Graphics*, 2002, vol. 26, issue: 5, pp. 745-751.

19. P. Reuter, I. Tobor, C. Schlick, and S. Dedieu, "Point-based Modelling and Rendering Using Radial Basis Functions", *Proceedings 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia*, 2003.

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

21. J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell, "Smooth Surface Reconstruction From Noisy Range Data", *ACM GRAPHITE 2003*, pp. 119-126.

22. S.-Y. Cho, and T.W.S. Chow, "Shape Recovery from Shading By a New Neural-Based Reflectance Model", *IEEE Transactions on Neural Networks*, 1990, vol. 10 issue 6, pp. 1536-1541.

23. S.-Y. Cho, and T.W.S. Chow, "A Neural-Learning-Based Reflectance Model for 3-D Shape Reconstruction", *IEEE Transactions on Industrial Electronics*, 2000, vol. 47, issue 6, pp. 1346 –1350.

24. D. Nandy, and J. Ben-Arie, "Shape from Recognition and Learning: Recovery of 3-D Face Shapes", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999, vol. 2.

25. M. Motegi, and Y. Kosugi, "Self-organizing Elastic Networks for Generating a 3D Model from Many Images", *Systems and Computers in Japan*, 1999, vol. 30, issue 12, pp. 106 – 115.

26. D.S. Chen, and R.C Jain, "Surface Reconstruction Using Robust Backpropagation", *IEEE International Conference on Neural Networks*, 1994, vol.6, pp. 4072 –4077.

27. E. Piperakis, and I. Kumazawa, "Affine Transformations of 3D Objects Represented with Neural Networks", Proceedings 3rd International Conference on 3-D Digital Imaging and Modeling, 2001, pp. 213 –223.

28. Y.-H. Tseng, J.-N. Hwang, and F.H. Sheehan, "3-D Heart Contour Delineation and Motion Tracking of Ultrasound Images Using Continuous Distance Transform Neural Networks", *Proceedings IEEE Signal Processing Society Workshop*, 1996, pp. 361 –370

29. Y.-H. Tseng, J.-N. Hwang, and F.H. Sheehan, "Three-dimensional Object Representation and Invariant Recognition Using Continuous Distance Transform Neural Networks", *IEEE Transactions on Neural Networks*, vol. 8, issue 1, 1997, pp. 141 –147.

30. A. Chella, and R. Pirrone, "A Two Stage Neural Architecture for Segmentation and Superquadrics Recovery from Range Data", in *Neural Nets,* Eds. M. Marinaro, R. Tagliaferri, *13th Italian Workshop on Neural Nets*, 2002, Lecture Notes in Computer Science 2486 Springer 2002, pp.132-139.

31. S.-W. Chen, G.C. Stockman, and K.-E. Chang; "SO Dynamic Deformation for Building of 3-D Models", *IEEE Transactions on Neural Networks*, 1996, vol. 7, issue 2, pp. 374 –387.

32. I. Kumazawa, "Compact and Parametric Shape Representation by a Tree of Sigmoid Functions for Automatic Shape Modeling", *Pattern Recognition Letters* 21, 2000, pp. 651-660

33. I. Kumazawa, M. Ohno, "3D Shape Reconstruction from Multiple Silhouettes: Generalization from Few Views by Neural Network Learning", in Eds. C. Arcelli et al., *International Workshop on Visual Form* 4, LNCS 2059, Springer Verlag, 2001, pp. 687-695

34. Y. Yu, "Surface Reconstruction From Unorganized Points Using Self-organizing Neural Networks", *Proceedings IEEE Visualization,* 1999, pp. 61 – 64

35. I. P. Ivrissimtzis, W-K. Jeong, and H-P. Seidel, "Using Growing Cell Structures for Surface Reconstruction", *Shape Modeling International*, 2003, p. 78

36. A. Saalbach, G. Heidemann, and H. Ritter, "Representing Object Manifolds By Parametrized SOMs", *Proceedings 16th International Conference on Pattern Recognition,* 2002, vol. 2, pp. 184 -187

37. M. Carcenac, and A. Acan, "Modeling an Isosurface with a Neural Network", *Proceedings 8th Pacific Conference on Computer Graphics and Applications*, 2000, pp. 165 –174

38. S. Loncaric, T. Markovinovic, T. Petrovic, D. Ramljak, and E. Sorantin, "Construction of Virtual Environment for Endoscopy", *IEEE International Conference on Multimedia Computing and Systems*, 1999, vol. 1, pp.: 475 –480.

39. I. Kumazawa, "Target Tracking By Matching a Shape Represented By a Tree of Sigmoid Functions", *Pattern Recognition Letters* 21, 2000, pp. 661-675.

40. S. Di Bona, and O. Salvetti, "An Efficient Method to Map a Regular Mesh into a 3D Neural Network", *Proceedings International Conference on Image Processing*, 2001, vol. 1, pp. 529 –532.

41. G. Coppini, R. Poli, and G. Valli, " Recovery of the 3-D Shape of the Left Ventricle from Echocardiographic Images", *IEEE Transactions on Medical Imaging*, 1995, vol. 14, pp. 301-317.

42. J.-N. Hwang, and Y.-H. Tseng, "Motion Estimation of Partially Viewed 3-D Objects Based on A Continuous Distance Transform Neural Network", *Proceedings IEEE International Conference Image Processing*, 1994, vol. 3, pp. 917 –921.

43. T. Chen, W-.C. Lin, and C-.T. Chen, "Artificial Neural Networks for 3D Nonrigid Motion Analysis", International Joint Conference on Neural Networks, 1992, vol. 4, pp. 420 –425.

44. T. Ishikawa, S. Morishima and D. Terzopoulos, "3D Face Expression Estimation and Generation from 2D Image Based on a Physically Constraint Model", *IEICE Transactions on Information and Systems*, 2000, vol. E83-D, no.2.

45. *The American Heritage Dictionary of the English Language*, [Online]. Available: http://www.bartleby.com/61/

46. R. Hall, "Supporting Complexity and Conceptual Design in Modeling Tools", in *State of the Art in Computer Graphics: Visualisation and Modeling*, Eds. D.F. Rogers, and R.A. Earnshaw, 1991 – quotation from R. Sproull, keynote speech, SIGGRAPH 1990

47. N. I. Badler, and A. S. Glassner, "3D Object Modeling", *SIGGRAPH 97 Introduction to Computer Graphics Course Notes*, 1997.

48. J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, *Computer Graphics. Principles and Practice*, Addison-Wesley, 1996

49. B. Fleming, *3D Modeling & Surfacing*, Morgan Kaufmann, 1999

50. A. Goodman, Lecture Notes SCC308 – Computer Graphics, Deakin University, Australia, [Online], Available: http://www.deakin.edu.au/~agoodman/scc308/topic4.html

51. T. Akenine-Moller, and E. Haines, *Real-Time Rendering*, A K Peters, 2002

52. M. E. Mortenson, *Geometric Modeling*, Second Edition, Wiley Computer Publishing, 1997

53. E. Angel, *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*, Third Edition, Pearson Education, 2003

54. J. Hoschek, and D. Lasser, *Fundamentals of Computer Aided Geometric Design*, A K Peters, 1993

55. H. J. Wolters, "Rational Techniques", in *Handbook of Computer Aided Geometric Design*, Eds. G. Farin, J. Hoschek, and M. S. Kim, Elsevier, 2002

56. G. Farin, *Curves and Surfaces for CAGD. A Practical Guide*, Fifth edition, Morgan Kaufmann, 2002

57. D. H. U. Kochanek, R. H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control", *Proceedings 11th annual International Conference on Computer Graphics and Interactive Techniques*, 1984, pp. 33 - 41

58. Eric Weisstein's World of Mathematics. A Wolfram Web Resource, [Online], Available: *http://mathworld.wolfram.com/*

59. G. Turk, and J. F. O'Brien, "Shape Transformation Using Variational Implicit Surfaces", *SIGGRAPH'99*, pp. 335–342

60. *, "Easily Achieving NURBs Patch Continuity", [Online]. Available: *http://www.swanimator.com/tipstutorials/patchModellilng/patchmodelling.htm.*

61. G. Yngve, and G. Turk, "Creating Smooth Implicit Surfaces from Polygonal Meshes", *Technical Report GIT-GVU-99-42*, Georgia Institute of Technology, 1999.

62. C.L. Bajaj, J. Chen, and G. Xu, "Modeling with Cubic A-Patches", *ACM Transactions on Computer Graphics*, 1995, vol.14, issue 2, pp. 103–133.

63. T. Karkanis, A. James Stewart, " Curvature Dependent Triangulation of Implicit Surfaces", *IEEE Computer Graphics and Applications*, 2001, vol. 22, no.2, pp. 60-69.

64. F. S. Hill, *Computer Graphics*, Macmillan, 1990

65. A.H. Barr, "Local and Global Deformations of Solid Primitives", *Computer Graphics*, 1984, vol. 18, nr.3.

66. F. Solina and R. Bajcsy, "Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations", *IEEE Tranactions on. Pattern Analysis and Machine Intelligence*, 1990, vol.12, issue 2, pp. 131–147.

67. R. J. Campbell, P. J. Flynn, "A Survey Of Freeform Object Representation and Recognition Techniques", *Computer Vision and Image Understanding*, 2001, vol. 81, no. 2, pp. 166-210.

68. A. G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems", *ACM Computing Surveys*, 1980, vol. 12, issue 4, pp. 437 - 464

69. V. Shapiro, "Solid Modeling", in *Handbook of Computer Aided Geometric Design*, Eds. G. Farin, J. Hoschek, and M. S. Kim, Elsevier, 2002

70. *, "Object-Centered Representations", [Online]. Available: *http://www.netnam.vn/unescocourse/computervision/832.htm*

71. H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys*, 1984, vol. 16, issue 2, pp.187-260.

72. *, "Point-Set Models of Solid", [Online]. Available: *http://weld.arc.cmu.edu/48-745/Lectures-handouts/*

73. D. Libes, "Modeling Dynamic Surfaces with Octrees", *National Institute of Standards and Technology*, 1991

74. J. Sinnott, and T.L.J. Howard, "SQUIDS: Interactive Deformation of Superquadrics for Model Matching in Virtual Environments", *Proceedings Eurographics*, 2000, pp. 73-80.

75. A. R. Smith, "Plants, Fractals, and Formal Languages", *Computer Graphics*, 1984, vol. 18, no. 3, pp. 1-9.

76. P. Prusinkiewicz, M. Hammel, R. Miech, and J. Hanan, "The Artificial Life of Plants", [Online]. Available: *http://www.siggraph.org/education/materials/HyperGraph/modeling/procedural_modeling/procedural_modeling.htm*

77. F. K. Musgrave, "Fractal Models of Natural Phenomena", *SIGGRAPH 01 "Simulating Nature" Course*, [Online], Available: *https://www.internal.pandromeda.com/engineering/musgrave/unsecure/S01_Course_Notes.html*

78. W.T. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects", *ACM SIGGRAPH Computer Graphics*, 1983, vol. 17, no.3

79. N. Sala, "Fractal Models in Architecture: A Case of Study", *Proceedings International Conference on "Mathematics for Living"*, 2000, pp. 266-272

80. Y. I. H. Parish, and P. Müller, "Procedural Modeling of Aities", *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* , 2001

81. T. McReynolds, D. Blythe, B. Grantham, and S. Nelson, Siggraph 99 *Advanced Graphics Programming Techniques Using OpenGL*, Course Notes, [Online]. Available: *http://www.opengl.org/developers/code/sig99/advanced99/notes/node335.html*

82. W. T. Reeves , R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", *ACM SIGGRAPH Computer Graphics*,1985, vol.19 nr.3, p.313-322.

83. C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1994

84. C. Jutten, "Supervised Composite Networks", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, pp. C1.6:1-13

85. M. Costa, P. Crispino, A. Hanomolo, and E. Pasero, "Artificial Neural Networks and the Simulation of Human Movements in CAD Environments", *International Conference on Neural Networks*, 1997, vol. 3, pp. 1781 -1784

86. J. N. Noyes, "Neural Network Training", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, B3

87. L. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*, Prentice Hall, 1994

88. P. J. Werbos, "Why Neural Networks", in *Handbook of Neural Computation,* Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, A2

89. E. Fiesler, "Topology" and "Fully Connected Topologies", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, B2.2:1

90. T. O. Jackson, "Data Input and Output Representations", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, B4

91. J.A. Sirat and J.P. Nadal, "Neural Trees: A New Tool for Classification", *Network*, 1990, vol.1, pp. 423-438.

92. S. Haykin, *Neural Networks. A Comprehensive Foundation*, Macmillan College Publishing Company, 1992

93. Martin F. Moller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", *Neural Networks*, 1993, vol. 6, pp. 525-533

94. M. Verleysen, "Feedforward models", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, C2.1

95. N. Davey, R.G. Adams, and S.J. George, "The Architecture and Performance of a Stochastic Competitive Evolutionary Neural Tree Network", *Applied Inteligence* 12, 2000, pp.75-93

96. T.M. Martinetz, S.G. Berkovich, and K.J. Schulten, "Neural-Gas Network for Vector Quantization and its Application to Time-Series Prediction", *IEEE Transactions on Neural Networks*, 1993, vol. 4, no. 4, pp.558-568

97. M. Daszykowski, B. Walczak, and D.L. Massart, "On the Optimal Partitioning of Data with K-means, Growing K-means, Neural Gas and Growing Neural Gas", *Journal of Chemical Information and Computer Sciences*, 2002, vol. 42, pp. 1378-1389

98. T. Kohonen, "The Self-Organizing Map", [Online], Available: *http://www.cis.hut.fi/projects /somtoolbox/theory/somalgorithm.shtml*

99. B. Fritzke, "Unsupervised Ontogenic Networks", in *Handbook of Neural Computation*, Eds. E. Fiesler, R. Beale, IOP Publishing Ltd and Oxford University Press, 1997, C2.4

100. N. Kasabov, *Evolving Connectionist Systems. Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines*, Springer Verlag, 2003

101. "SOM Toolbox Online Documentation", [Online], Available: *http://www.cis.hut.fi /projects /somtoolbox/documentation/*

102. A. M. Cretu, E. M. Petriu, G. G. Patry, "Neural Network Architecture for 3D Object Representation", *Proceedings of the HAVE 2003 - IEEE International. Workshop on Haptic, Audio and Visual Environments and their Applications*, 2003, pp. 31-36, Ottawa, ON, Canada