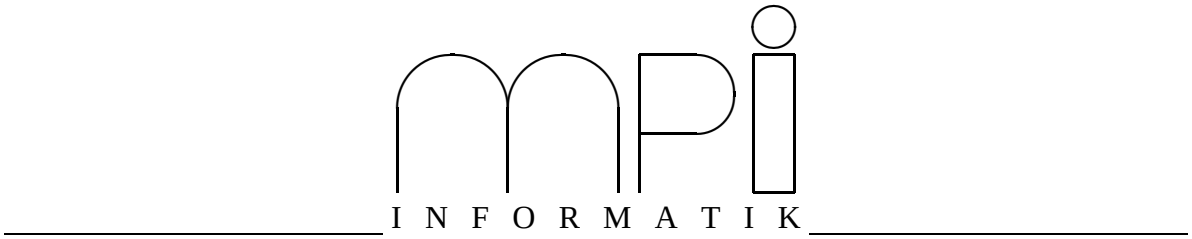



# Neural Meshes: Statistical Learning Methods in Surface Reconstruction

I.P. Ivrissimtzis W-K. Jeong H-P. Seidel

MPI–I–2003–4–007 April 2003

FORSCHUNGSBERICHT RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Stuhlsatzenhausweg 85 66123 Saarbrücken Germany

**Authors' Addresses**

Ioannis Ivrissimtzis
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
ivrissim@mpi-sb.mpg.de

Won-Ki Jeong
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
jeong@mpi-sb.mpg.de

Hans-Peter Seidel
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
hpseidel@mpi-sb.mpg.de

**Abstract**

We propose a new surface reconstruction algorithm based on an incrementally expanding neural network known as Growing Cell Structure. The neural network learns a probability space, which represents the surface for reconstruction, through a competitive learning process. The topology is learned through statistics based operations which create boundaries and merge them to create handles. We study the algorithm theoretically, calculating its complexity, using probabilistic arguments to find relationships between the parameters, and finally, running statistical experiments to optimize the parameters.

**Keywords**

Statistical Learning, Neural Networks, Growing Cell Structures, Surface Reconstruction

# 1 Introduction

The neural networks and the statistical learning methods were first introduced as theoretical concepts in the late 40's. But it was the last 20 years, with the rapid increase of computer's speed, that we witnessed an explosion in the application side of these powerful methods. A variety of problems, ranging from modeling gene sequences to handwriting recognition, have been studied with statistical learning methods, sometimes with more sometimes with less success.

In a typical situation where statistical learning methods can be employed, we have a set of observations, statistical data or results from experiments, and we want to fit a mathematical model on them with criteria ranging from fairness to the ability to predict some future observations. From this point of view, the problem of surface reconstruction from a point cloud, usually obtained from a 3D scanner or a satellite, is particularly well-suited to be studied with statistical learning methods. Nevertheless, and despite some classic papers like [21] which has shown the potential of a signal-theoretic approach, the majority of the proposed methods is still geometry oriented. That means that the data are not interpreted as a set of signals from the surface to be reconstructed, but rather as a part of the surface which has to be processed to give the model describing the surface.

In this paper we propose a statistical learning algorithm for generating a triangle mesh from a point set, which can either be a point cloud, or a surface described implicitly or even another polygonal mesh. Instead of processing directly this point set we start with an initial mesh, usually a tetrahedron, thought here as an initial guess for the surface, and we process it according to signals obtained by randomly sampling our point set. We call the triangle meshes we construct with this method *Neural Meshes*, although at first glance they only have a remote resemblance with the most well-known types of neural networks. The reason is that our algorithm is inspired, and heavily influenced, from Fritzke's *Growing Cell Structures* [7] which are considered as a special type of *Neural Networks*.

In a typical response to a signal, the algorithm processes the neural mesh by finding the vertex nearest to the signal and moving it towards the signal. Then it smoothes the neighborhood of this best matching vertex. The fact that only the vertex nearest to the signal moves towards it, traditionally, is interpreted as a kind of competition between the vertices of the neural mesh to adapt to the signal, and such a process is called *competitive learning*. Except of this basic learning step, other operations like vertex split, half edge collapse, triangle removal and boundary merging, based on the combination of an evaluation of the recent activity of each vertex and a statistical analysis of the mesh, ensure that the mesh grows and adapts to the geometry and topology of the target space.

## 1.1 Related Work

The Growing Cell Structures [7], are neural networks trained in a competitive learning process to model an unknown probability space $\mathscr{P}$. We start with an initial simplicial complex with the vertices, thought as the nodes of the network $\mathscr{N}$, carrying geometric information. Unlike some other types of neural networks, there are no weights attached to the edges which carry connectivity information only. The training is competitive and $\mathscr{N}$ grows incrementally by splits of the most active vertices. The activity of a vertex is a measure reflecting how many times a particular vertex has been the one nearest to the sample, with the most recent activity counting more.

The main difference between the Growing Cell Structures and the previously proposed and more popular Self-Organizing Maps (SOM's) [17] is that they grow incrementally, inserting one new vertex after the other. In many applications this is a crucial difference [9], while in surface reconstruction in particular it offers the necessary flexibility we need to learn concavities and other surface features. A more detailed introduction to Growing Cell Structures can be found in [5], a classic introduction to neural networks can be found in [4], while a comprehensive introduction in the more general framework of statistical learning can be found in [12].

Earlier work with some similar techniques employed in Geometric Modeling and Visualization related problems, include [11] where SOM's are used for the visualization of multi-dimensional data, [13] where SOM's are used for Free-form surface reconstruction, [25] where Growing Cell Structures are used for the same purpose, [26] where SOM's are used in mesh generation, [3] where SOM's are used for grid fitting, and [15] where a technique similar to the one proposed here is used for the reconstruction of a closed surface of genus 0.

The physical models is another example of techniques similar to ours. Like many other Neural Networks applications which are also inspired from Physics based methods, the Neural Meshes are conceptually similar to the snakes and the active surfaces [23], [24], [20]. Indeed, in the Basic Step of the algorithm the processes of geometry learning and smoothing can be thought as an external and an internal force, respectively, applied to the Neural Mesh.

On the other side, there are many, well-established, geometry oriented techniques, proposing innovative solutions to the surface reconstruction problem. Mentioning only papers nearest to a Computer Graphics approach: [14] calculates approximating tangent planes and uses volumetric methods to construct a triangle mesh, [2] uses volumetric techniques on $\alpha$-shapes to produce a piecewise polynomial surface, [18] fits a B-Spline surface to the data and then calculates detail vector displacements, [1] uses 3D Voronoi diagrams, [22] uses density scaled $\alpha$-shapes, [16] simulates the wrapping of a plastic membrane around the object, [6] interpolates points with normals with polyharmonic RBF's, [10] solves a dynamical system over a distance function obtained from the sample.

Compared to these methods Neural Meshes have a different philosophy, probably better suited to a machine, as they are based on the repetition of a very simple procedure. At a more practical level, the main advantages of our algorithm is that it only samples the data set and does not process it, and thus, its performance is independent of the size of the data set, and secondly, that it reconstructs the surface Down to Top and a coarse approximation of any data set can be obtained immediately. Another attractive feature of the algorithm is the absence of topological noise, that is, tiny boundaries or handles caused for example by misaligned range images. Notice the reconstruction of the David model at the end of the paper which started from topologically noisy data.

## 1.2 Overview

The algorithm starts with a probability space $\mathscr{P}$ which is repeatedly sampled, and an initial neural mesh $\mathscr{M}$ which is processed according to the samples. For a sample $s$ from $\mathscr{P}$ we find the vertex $v_w$ of $\mathscr{M}$ which is nearest to $s$, traditionally called the *winner*, and move it towards $s$. Then the 1-ring neighborhood of $v_w$ is smoothed by applying a Laplacian smoothing operator acting on the tangential direction. Similarly to [7], the activity of each vertex is measured with a signal counter attached to it, and the most active vertices are duplicated with a vertex split, while the least active vertices

are removed with a half edge collapse. The idea behind it is that the activity of a vertex is a measure of how important role it plays in the representation of $\mathscr{P}$ by $\mathscr{M}$ and thus, we duplicate the most important vertices while we remove the least important.

Then, we have two topology changing operations. The first is triangle removal, where triangles with area larger than a threshold are removed creating a boundary. The threshold is calculated using the mean average of the areas of the triangles of $\mathscr{M}$. The justification of this step, as we will see later, is that the area of a triangle is inversely analogous to the density of $\mathscr{P}$ near this triangle, and therefore the very large triangles represent parts of $\mathscr{P}$ which should not be represented at all. The second topology changing operation merges two boundaries with Hausdorff distance below a threshold creating a handle.

The rest of the paper is organized as following: after a brief discussion of the differences between the present paper and [7] and [15], in Section 2 we present the algorithm in more detail. In Section 3 we analyze the algorithm, explaining heuristically why it works, and using probabilistic arguments and statistical experiments to find an optimal set of parameters. In Section 4 we present some results and discuss some special applications, and we conclude with a brief discussion of our future work.

## 1.3 Our Contribution

The original Growing Cell Structures were not designed for triangle meshes specifically but for $n$-dimensional simplicial complices. Having some obvious advantages this generality also imposes several restrictions. For example the networks grow with edge splits, an operation which can easily be generalized to arbitrary dimension, while the most inactive vertices are simply removed, changing sometimes topology. Here, as we deal with triangle meshes only, we use vertex splits and half edge collapses respectively, see [14], which, we have found, give better connectivities. Also, as there are simple criteria checking if a half edge collapse changes the mesh topology we can use such an operation more frequently, improving further the quality of the final mesh.

The other main difference with [7] is in the way the neighbors of the winner adapt to the signal. The Growing Cell Structures and many other competitive learning methods follow the general idea that not only the winner should respond to a signal, but, for stability reasons and faster convergence of the network, the neighbors of the winner should also learn from the signal although at a smaller rate. Here, as we have a 2D structure embedded in the 3D space, we can use instead smoothing operators in the tangential direction which do not interfere with the learning process. That is, the general philosophy of the learning step of our algorithm can be described as: The winner learns from the signal and then its neighborhood is smoothed.

Going to the main differences with [15], the first is that now the neural network learns the topology. That learning involves procedures to remove triangles and create boundaries, merge boundaries and create handles, as well as the handling of boundary vertices as special cases in the vertex split and half edge collapse operations. The main difference from other topology learning algorithms, as for example [19], [8], is that now the main primitives of the process are the boundaries rather than the vertices and the edges. In practice this reduces the topological noise considerably.

The second main difference is in the analysis of the algorithm which now is more mathematical. The description of the algorithm involves many parameters and their evaluation in [15] was empirical. That is, we were experimenting with different values and then we visually inspected the resulting model to check for convergence, convergence to local minima, or fold-overs. Here we quantify the mesh quality, using the

simplest possible criterion, that is, the ratio of the valence 6 vertices in the mesh, and then we run statistical experiments to optimize the parameters. Notice that this is an intrinsic metric of the mesh quality. It measures the Neural Mesh itself without comparing it against any target space. Using instead an external geometric measure it would be out of the spirit of the paper as the target is considered unknown.

# 2 Neural Meshes

The basic input of the algorithm is a probability space $\mathscr{P}$ which is sampled, returning one point at the time, and an initial mesh $\mathscr{M}$ which at each step is processed by the algorithm according to the sample. The probability space $\mathscr{P}$ is thought here as representing the surface we want to reconstruct.

The algorithm can be described in the following steps:

1. **Basic Step**
   - Sample the target space $\mathscr{P}$ and return one point $s$.
   - Find the vertex $v_w$ of $\mathscr{M}$ which is nearest to $s$.
   - Update the position of $v_w$ by
   $$v'_w = (1-\alpha_w)v_w + \alpha_w s \tag{1}$$
   where $\alpha_w$ is a constant.
   - Apply $C_L$ iterations of Laplacian smoothing, in the tangential direction, on the 1-ring neighborhood of $v_w$, with parameter $\alpha_L$, where $C_L, \alpha_L$ are constants.
2. **Vertex Split:** After a constant number $C_{vs}$ of iterations of the Basic Step, calculate the signal counter of each vertex, and split the vertex with the highest signal counter. The signal counter is a real number measuring the activity of each vertex.
3. **Half Edge Collapse:** After a number of iterations of the Basic Step, find the least active vertices from the history list of recent activity and remove them with a half edge collapse.
4. **Triangle Removal:** After a number of iterations of the Basic Step remove the triangles with area larger than
   $$\alpha_r \mathrm{E(Area)} \tag{2}$$
   where E(Area) is the mean area of the triangles of $\mathscr{M}$, and $\alpha_r$ is a constant.
5. **Boundary Merging:** After a number of iterations of the Basic Step merge the boundaries with Hausdorff distance less than
   $$\alpha_m \sqrt{E(Area)} \tag{3}$$
   where $\alpha_m$ is a constant.
6. The algorithm terminates when some criteria are satisfied, e.g. a certain number of vertices has been reached.

Next, we discuss each step of the algorithm in more detail.

## 2.1 Basic Step: Geometry Learning

With the Basic Step of the algorithm the vertices of the Neural Mesh converge towards $\mathscr{P}$. It consists of four simpler steps. First, the target space $\mathscr{P}$ is sampled and a single point signal $s$ is returned. The underlying set $\Omega$ of $\mathscr{P}$ is usually a point cloud and the probability distribution is usually the discrete uniform. Nevertheless, we also deal with non-uniform distributions, while in some examples the underlying set $\Omega$ is a surface described implicitly. Notice that this is the only step in the whole algorithm that the target space is involved, making the speed of the algorithm practically independent from the size of the target space.

Then the neural mesh $\mathscr{M}$ is processed adapting to the signal $s$. We find the vertex $v_w$ of $\mathscr{M}$ which is nearest to $s$. The search is done using an octree updated every time the position of a vertex changes. The vertex $v_w$ is called the winner and adapts its position moving towards $s$, as shown by Eq. 1, learning this way the geometry of $\mathscr{P}$. The fact that only the winner is learning classifies our algorithm into the category of competitive learning algorithms.

The Basic Step continues with $C_L$ consecutive iterations of local Laplacian smoothing, in the tangential direction, with parameter $\alpha_L$. In all our applications $C_L = 5$. The Laplacian smoothing is essential to prevent and resolve unwanted foldovers. In the original Growing Cell Structures, this step is much simpler with the 1-ring neighbor of $v_w$ also moving towards the sample but at a lesser rate than $v_w$ itself. The natural interpretation is that for stability and faster convergence the neighborhood of $v_w$ is also learning from $s$. Here the philosophy is different and only the winner learns from $s$, while its neighborhood is smoothed. Our experiments have shown the Laplacian smoothing in the tangential direction to be neutral regarding the geometry learning, and that allows us to choose $\alpha_L$ from a very large range of possible values. Indeed, the $\alpha_L$ can be even 10 times larger than the $\alpha_w$ without affecting the rate of convergence to the right geometry.

However, the balance between $\alpha_w$ and $\alpha_L$ is still crucial for the optimal performance of the algorithm and will be studied more systematically in Subsection 3.4.

## 2.2 Connectivity Changes: Steps 2-3

The Basic Step does not change the connectivity of the neural mesh. This is done in Steps 2 and 3.

The Step 2 is called every $C_{vs}$ iterations of the Basic Step, where $C_{vs}$ is an integer constant. A typical value for $C_{vs}$ can range between 50 and 1000. It increments the number of vertices of the neural mesh, by splitting the vertex with the highest activity as it is measured by the signal counter. Following [7], in order to calculate the signal counter we use a constant $\alpha_{sc}$, and at each iteration of the Basic Step we add 1 to the signal counter $c_i$ of the winner and then multiply the signal counter of all the vertices with $\alpha_{sc}$. A typical value for $\alpha_{sc}$ is 0.95. Notice that the recent activity of a vertex weights more on the signal counter.

To speed up the algorithm the evaluation of the signal counter is done only when it is needed, that is, every time the Step 2 is called. Between two calls we store the indices of the winners and then multiply the signal counter of each vertex $v_i$ by

$$\alpha_{sc}^{C_{vs}}(1+\alpha_{sc}^{-x_1}+\alpha_{sc}^{-x_2}+\cdots+\alpha_{sc}^{-x_j}) \tag{4}$$

where $x_i,\ldots,x_j \in [1...C_{vs}]$ are the iterations between two calls of Step 2 for which $v_i$ was the winner. Notice that at most $C_{vs}$ signal counters are multiplied with a number

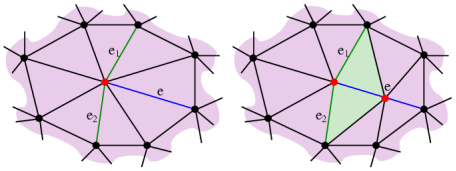



Figure 1: Vertex split.

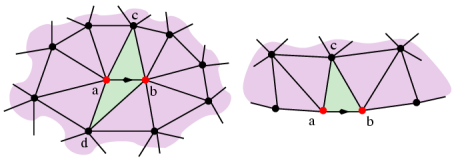



Figure 2: Half edge collapse.

different than $\alpha_{sc}^{Cvs}$.

After the vertex $v_i$ with the highest signal counter is calculated we find the longest edge $e$ with one end at $v_i$ and traverse both directions from $e$ equally to find two edges $e_1, e_2$, neighboring $v_i$, such that $e_1, e_2$ split the star of $v_i$ approximately at half. We split along $e_1, e_2$ distributing this way the valences as regularly as possible. The new vertex is placed in the middle of $e$, see Fig. 1.

The signal counter of the split vertex is divided between the two new vertices in a ratio corresponding to the area of their *restricted Voronoi cells*, that is, the intersection of their Voronoi cells with the surface that has to be learned. The reason for this choice will be apparent after the discussion in Subsection 3.1. In our implementation we replace the area of the restricted Voronoi cell of a vertex $v$ with the area $F_v$ of a square, given by

$$F_v = (l_v)^2 \tag{5}$$

where

$$l_v = \frac{1}{\text{valence}(v)} \sum_{v_i \in \text{1-ring}(v)} \|v_i - v\| \tag{6}$$

as in [7].

For the Step 3 we have to identify the least active vertices and remove them with a half edge collapse. In theory we could use again the signal counter but we would soon run into numerical instabilities. Indeed, let $\sigma$ be the sum of all the signal counters. After one iteration we have

$$\sigma' = \alpha_{sc}(\sigma + 1) \tag{7}$$

and this number tends to

$$\frac{1}{1 - \alpha_{sc}} \tag{8}$$

Also, if $c_M$ is the largest number such that

$$\frac{1}{1 - \alpha_{sc}} \alpha^{c_M} > 0 \tag{9}$$

in machine accuracy, then any vertex which has not been the winner for the last $c_M$ iterations has signal counter equal to 0. In particular there can be up to $c_M$ vertices with counter greater than 0. In our system $c_M \simeq 2000$ and that means that we have to increase the machine accuracy considerably to find the most inactive vertices of a large neural mesh with the use of the signal counter.

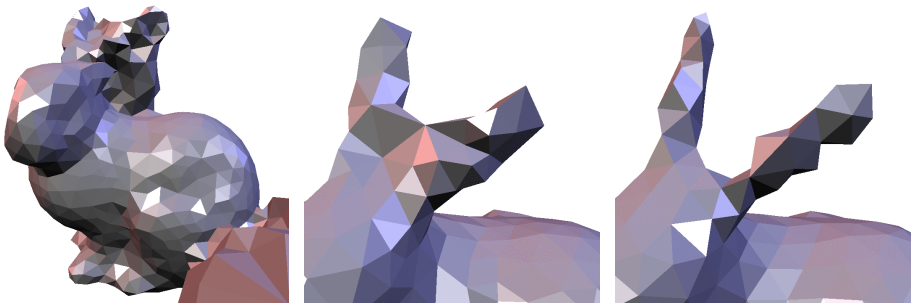


Figure 3: Left: The half edge collapse is inactivated ($C_{ec} = \infty$). Middle: $C_{ec} = 50$. Right: $C_{ec} = 5$.

Instead, at every call of the Step 3 we find the vertices that have not been the winners since the last call of the same Step 3 and remove them with a half edge collapse. This simple solution works as long as the frequency of the Step 3 takes into account the current number of vertices of the neural mesh. Thus, we call Step 3 every $C_{ec}v$ iterations of the Basic Step where $v$ is the number of vertices of the neural mesh at the previous call and $C_{ec}$ a constant.

When a vertex is selected to be removed we collapse it towards one of its neighbors chosen in a way that minimizes the connectivity irregularity measure given by

$$\frac{1}{3}\sqrt{(a+b-10)^2+(c-7)^2+(d-7)^2} \tag{10}$$

for inner vertices, and

$$\frac{1}{2}\sqrt{(a+b-7)^2+(c-7)^2} \tag{11}$$

for boundary vertices, see Fig. 2. By checking the legality of every half edge collapse before performing it, we make sure that the Neural Mesh remains a manifold throughout the learning process. An additional requirement we impose is that we do not split a boundary vertex with valence less or equal to 4 to avoid a proliferation of vertices with negative irregularities on the boundary.

The Step 3 is instrumental for the quality of the neural mesh and it functions in a many-fold way. First, it removes the misplaced vertices of the neural mesh because such vertices will never be the winners and at some stage will be removed. In particular, Step 3 resolves the situations where the neural mesh converges to a local minimum, a common problem with the neural network convergence. Also, from Fig. 3 we can see that Step 3 plays a role in the learning of the concavities of $\mathscr{P}$.

Another reason for a vertex to be inactive for long time is because it is located in a part of the neural mesh which over-represents $\mathscr{P}$ or in other words, it is located in an area where the competition for learning from a sample is very high. By removing such vertices the representation of $\mathscr{P}$ by the neural mesh becomes fairer.

## 2.3 Topology changes: Steps 4-5

The Steps 4,5 change the mesh topology using some simple statistical criteria which, in principle, can be checked as often as we wish. But, as they are time consuming, it

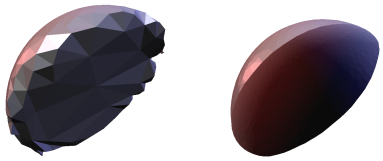


Figure 4: Triangle removal.

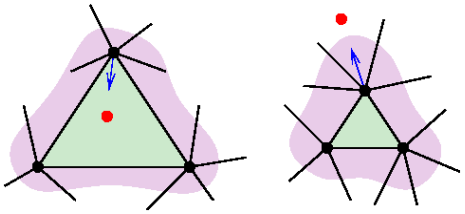


Figure 5: The movement of the winner tends to make triangles with equal probability measure.

is better to call these steps rarely and with decreasing frequency. The simplest solution that will also simplify the algorithm is to call them successively, immediately after each call of Step 3.

The justification of Step 4 is that, as we will see in Section 3, the triangles of a neural mesh tend to have area inversely analogous to the density of the probability at the part of $\mathscr{P}$ they represent. As a consequence, very large triangles represent parts of $\mathscr{P}$ with very low probability density. When this density is below a certain threshold we can consider it insignificant and remove the corresponding triangles. Fig. 4 (Right) shows the large triangles on the great disk of the hemisphere that will be removed by Step 4, creating a boundary. It also gives a more intuitive idea on why the large triangles can be considered as misplaced and should be removed. Similarly to the half edge collapse of Step 3, we first check the legality of a triangle removal before performing it, making sure that the Neural Mesh remains a manifold throughout the learning process.

Finally the Step 5 merges two boundaries when they are relatively near to each other, creating this way handles. In our implementation the Hausdorff distance between the boundaries is approximated by the Hausdorff distance of the vertex sets of these boundaries.

Since the two merging boundaries are very close to each other, we can use a simple tiling method to merge them. We start with the two closest vertices, one from each boundary, and traverse both boundaries in the same orientation checking all the possible triangles we can construct with the next vertices. Among them, we choose the best triangle and add it to the neural mesh, repeating the procedure until the boundaries are completely connected with a set of triangles. In our implementation the criterion of how good is a triangle depends on how close it is to be equilateral.

Notice that even though we assume that the two boundaries are close enough, still, sometimes we might create foldovers as we connect them. Indeed, while a real boundary will always slightly underestimate $\mathscr{P}$, because otherwise there will be no signals outside it and will shrink, nevertheless, two boundaries that are about to merge will have signals from both sides and may overlap. We let these foldovers to be resolved later in the process with the Laplacian smoothing of the Basic Step.

For some examples see the reconstruction of the Eight and the David model in the last page.

# 3 Analysis

## 3.1 Heuristics

The geometry of $\mathscr{P}$ is learned by the neural mesh $\mathscr{M}$ with repeated iterations of the Basic Step. After many iterations of it, and for a large enough neural mesh we may assume that $\mathscr{M}$ is close enough to $\mathscr{P}$ and therefore, that the probability space $\mathscr{P}$ induces an approximating *probability measure* on $\mathscr{M}$. In the core of the algorithm lies the argument that the density of the vertices of $\mathscr{M}$ reflects this probability measure. That is, we have a larger concentration of vertices near the parts of $\mathscr{P}$ where the distribution is denser.

To see this we assume the neural mesh $\mathscr{M}$ at a state of convergence. That means that the restricted Voronoi cells corresponding to the vertices of $\mathscr{M}$ tend to have equal measures. Indeed, as the probability for a vertex to be the winner is equal to the probability measure of its restricted Voronoi cell, the vertices with restricted Voronoi cells of large measure are more likely to split, while these with restricted Voronoi cells of small measure are more likely to collapse. By this process the restricted Voronoi cells tend towards equal probability measure bringing $\mathscr{M}$ nearer to the state of convergence.

Notice that the above is a global argument. At a local level we may assume that the distribution of $\mathscr{P}$ is uniform, which means that the probability measure is equal to the area measure of the neural mesh. In this case we can argue that the triangles $\mathscr{M}$ tend to have equal probability measure and area. Notice that being at a local level now, the heuristics should take into account the movement of the vertices at the Basic Step rather than vertex splits and half edge collapses.

Indeed, the move of the winner towards the sample tends to create triangles with equal probability measure, which locally means triangles of equal area. Indeed, let the winner be a vertex $v_i$ of a triangle $T$. If the probability measure of $T$ is large then the $v_i$ will be, with high probability, in the interior of $T$ which as a result will shrink, while if the probability measure of $T$ is small then $v_i$ will be, with high probability, outside $T$ which as a result will expand, see Fig. 5. Moreover, the Laplacian smoothing also tends to create triangles of equal area and thus of equal probability measure.

The above observation has also implications on the quality of the connectivity of $\mathscr{M}$. For a given arbitrary state of $\mathscr{M}$, the vertices of high valence usually have larger restricted Voronoi cells. That means that they are more likely to split during the process, improving this way the connectivity. Another consequence is that, as we have already mentioned, the triangles with very large area represent parts of $\mathscr{P}$ with very thin probability distribution, and when this density passes below a threshold it justifies the removal of the corresponding triangles.

The above heuristics can also be verified experimentally. See for example Fig. 8 (Right) where two implicit models where sampled non-uniformly.

## 3.2 The expansion rate of a neural mesh

As the algorithm both inserts new vertices in $\mathscr{M}$ (Step 2) and removes existing ones (Step 3), a naturally arising question is the relative rate $r_e$ of vertex insertion and removal. In other words, how fast does the mesh expand or shrink.

In a state of convergence the probability measures of the restricted Voronoi cells tend to be equal. The probability of a vertex to be the winner is equal to the probability measure of its restricted Voronoi cell, therefore also tends towards the Discrete Uniform distribution. Hence, the probability of a vertex to be the winner exactly $k$ times for $C_{ec}v$

iterations of the Basic Step (that is, between two calls of Step 3) follows the Poisson distribution $p(k;\mu)$, where

$$P(k;\mu) = e^{-\mu}\frac{\mu^k}{k!} \quad \text{with} \quad \mu = \frac{vC_{ec}}{v} = C_{ec} \tag{12}$$

The probability that a vertex was never the winner and has to be removed is

$$p(0;C_{ec}) = e^{-C_{ec}}. \tag{13}$$

Therefore, because of the linearity of the Expected Values Functions we expect

$$ve^{-C_{ec}} \tag{14}$$

vertices to be removed, while at the same time $vC_{ec}/C_{vs}$ new vertices are introduced with vertex splits. The expansion ratio of the neural mesh is given by

$$r_e = \frac{vC_{ec}/C_{vs}}{ve^{-C_{ec}}} = \frac{C_{ec}}{C_{vs}e^{-C_{ec}}} \tag{15}$$

and a necessary condition for the neural mesh to expand rather than shrink is that $r_e > 1$, giving

$$C_{ec} > e^{-C_{ec}}C_{vs} \tag{16}$$

Eq. 16 gives a lower bound necessary for the expansion of the neural mesh but does not guarantee expansion because it assumes the neural mesh at a stage of convergence. However, in practice we have found that a value of $C_{ec}$ greater than the bound of Eq. 16 by 1 or 2 will give an expanding neural mesh. Also, notice that the above problem, essentially, is a reformulation of the classic "Bins and Balls" problem in stochastic analysis, where $n$ balls are randomly distributed into $k$ bins.

## 3.3 Theoretical Complexity and time Performance

Here we calculate the theoretical complexity of each part of the algorithm under some reasonable assumptions. We will only outline the arguments because in practice, for meshes up to few tens of thousands triangles, the performance of the algorithm is still dominated by some constant multiplicative factors.

The Basic Step is repeated $O(v)$ times, the search in the octree for the winner and the updating of the octree are $O(\log v)$ while the Laplacian smoothing is constant in time. Therefore the total complexity of the Basic Step is $O(v\log v)$.

The Step 2 is repeated $O(v)$ times and while the vertex split is constant in time, the calculation of the signal counter is $O(v)$ making the total complexity of Step 2 quadratic. The Steps 3,4,5 are repeated $O(\log v)$ times. By Eq. 14 Step 3 is $O(v)$, while Step 4, involving the calculation of the mean area of the mesh triangles, is also $O(v)$. Finally, assuming that a boundary, being 1-dimensional, has $O(\sqrt{v})$ vertices, the complexity of Step 5 is also $O(v)$, giving a total complexity for the Steps 3,4,5 equal to $O(v\log v)$.

Table 1 shows the times achieved on a 1.7GHz PC for neural meshes of different size and $C_{vs} = 100,\ C_{ec} = 10$. An overlap in the Octree and Smooth loops makes the total less than the sum of the components. Although it is clear that the performance is dominated by the constant multiplicative factors we can also verify the above theoretical analysis.

Table 1: Timings.

| #v | Octree | Counter | Smooth | Steps 3-4-5 | Total |
|---|---|---|---|---|---|
| 1K | 15s | 4s | 77s | 0s | 86s |
| 2K | 34s | 17s | 159s | 1s | 190s |
| 5K | 96s | 117s | 430s | 3s | 10min |
| 10K | 213s | 456s | 884s | 10s | 25min |
| 20K | 446s | 1884s | 1832s | 36s | 74min |

We notice that the rather trivial calculation of the signal counter makes the complexity of the algorithm quadratic, and indeed for large meshes decreases considerably the performance. One remedy is to keep and update the list of the non-zero (in machine accuracy) signal counters and calculate the new signal counters only for them. Then, Step 2 will become $O(v)$ and the whole algorithm will be $O(v \log v)$.

But the above solution does not address the whole problem of unnecessary calculations. Indeed, for very large neural meshes the majority of the calculations becomes unnecessary as the mesh reaches a state of convergence. A simple solution is to split more than one vertex at each call of Step 2, and this will improve the time performance with only negligible deterioration of the mesh quality.

## 3.4 Statistical Analysis

Next we outline a methodology for finding optimal values for the main parameters $\alpha_w$, $\alpha_L$, and $C_{vs}$, $C_{ec}$. As we saw in 3.2 the parameter $C_{ec}$ is closely related to $C_{vs}$, and both affect the speed of the algorithm. Therefore, for simplicity, and given that time is always a prime consideration which can impose strict limits, we study the relation of $\alpha_w$, $\alpha_L$ separately, and then we see how the choice of $C_{vs}$ the affects the mesh quality.

For the choice of the parameters $\alpha_w$, $\alpha_L$ our first consideration is the geometric convergence. Notice that it is always possible to find a pair of values that will guarantee geometric convergence, given that for $\alpha_L = 0$ the neural mesh will converge, and the question is on the range of the acceptable values for $\alpha_L$. The second consideration is to avoid foldovers and other unwanted behavior as convergence to the wrong topology.

In [15] the methodology was to experiment with various pairs of $\alpha_w, \alpha_L$, each time inspecting visually the final neural mesh for any deficiencies, and thus, finding empirically an acceptable range for the parameters. Here, we outline a more systematic method, based on the quantification of the mesh quality by the ratio of the valence 6 vertices.

To set up the experiment we first selected a representative set of target spaces. It contains some well known models which we use here as point clouds. For the fine-tuning of the parameters it is important to include some less smooth models, here the Hand, because the algorithm behaves exceptionally well with the usual smooth models. We set $C_{vs} = 100$, $C_{ec} = 10$ and we run the algorithm 20 times to obtain a statistically significant number of observations, recording each time the average percentage of valence 6 vertices.

The first two diagrams in Fig. 6 show the behavior of the method as the rate of Laplacian smoothing increases, for $\alpha_w = 0.06$ and 0.12, respectively. It is clear that the quality of the mesh first increases with $\alpha_L$ but at some point starts to decrease for the

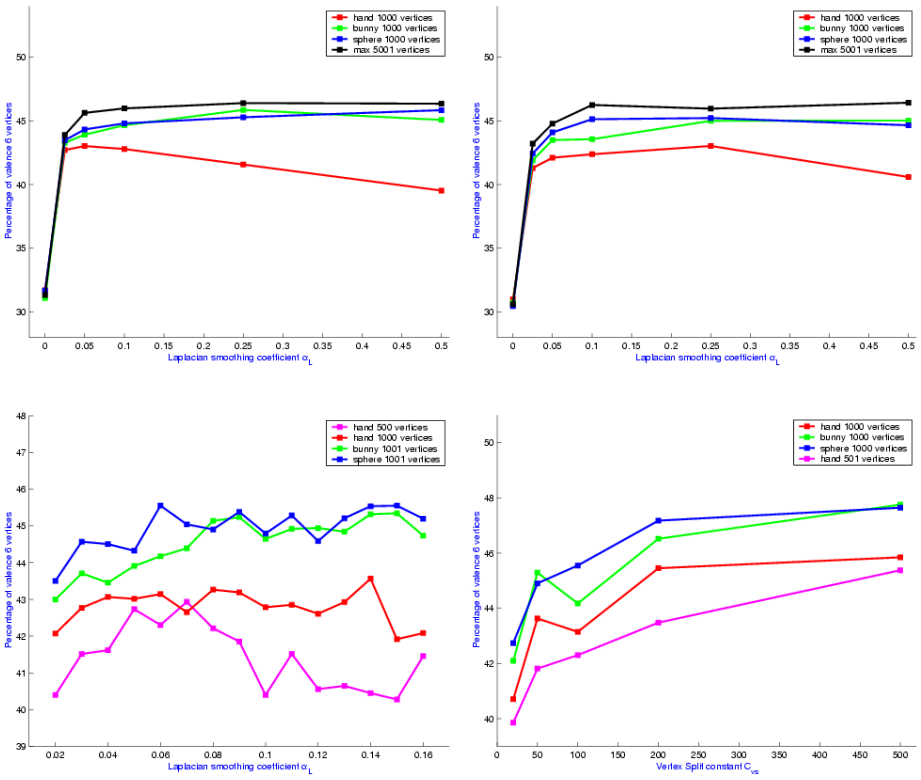



Figure 6: Up: The percentage of regular vertices for $\alpha_w = 0.6$ (Left), and $\alpha_w = 0.12$ (Right), as a function of $\alpha_L$. Down: The percentage of regular vertices for $\alpha_w = 0.06$ as a function of $\alpha_L$ (Left), and for $\alpha_w = 0.06, \alpha_L = 0.06$ as a function of $C_{vs}$ (Right). Each node represents the mean average of 20 experiments.

non-smooth models. The third diagram sets $\alpha_w = 0.06$ and shows this effect in finer detail. We notice that for values of $\alpha_L$ around 0.06 the shape of $\mathscr{P}$ and the size of the neural mesh are less significant for the quality of the connectivity. The final diagram sets $\alpha_w = 0.06,\ \alpha_L = 0.06$ and shows that mesh quality increases with $C_{vs}$.

# 4 Results

At the end of the paper we show some reconstructed neural meshes. We set $\alpha_w = 0.06,\ \alpha_L = 0.06$ and $C_{vs} = 100,\ C_{ec} = 10$, which is a reasonable trade-off between quality and speed. The other two parameters were set at $a_r = 7,\ a_m = 5$. For the David model in particular, up to the 10k vertex we used $C_{vs} = 500,\ C_{ec} = 8$. This way the neural mesh learned faster the concavities of the model and we were able to recover the correct topology at an early stage.

Fig. 8 shows some typical wireframe views of neural meshes. A characteristic semi-regular pattern is detectable and it was repeated throughout all our experiments. Table 2 shows the frequency of each valence for some reconstructed models. Notice that the distribution of the valences is independent from the shape of the surface and

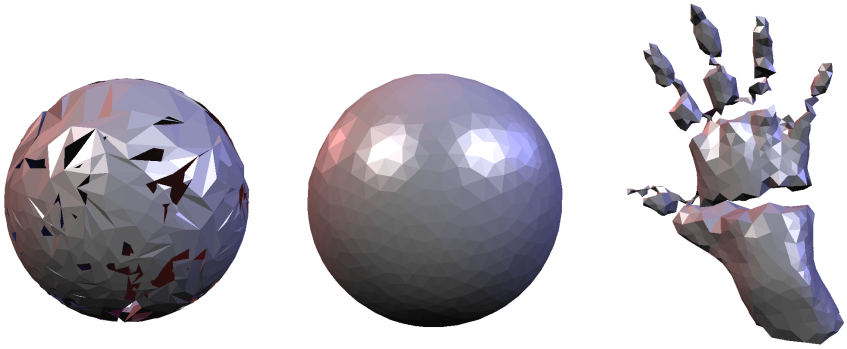


Figure 7: For $\alpha_w = 0.06$. Left: $\alpha_L = 0$. Middle and Right: $\alpha_L = 0.5$.

the size of the neural mesh.

Table 2: Valence distribution for some typical meshes.

| Model | Valence | | | | | | |
|---|---|---|---|---|---|---|---|
| | 4 | 5 | 6 | 7 | 8 | 9 | other |
| Bunny 1k | 1.2 | 28.4 | 47.3 | 17.5 | 5.0 | 0.7 | 0.0 |
| Bunny 5k | 1.0 | 28.1 | 46.4 | 19.9 | 3.7 | 0.6 | 0.2 |
| Bunny 20k | 1.1 | 28.4 | 46.0 | 19.7 | 4.0 | 0.6 | 0.3 |
| Dino 2k | 1.6 | 28.3 | 44.8 | 20.6 | 4.0 | 0.5 | 0.2 |
| Dino 20k | 0.8 | 29.7 | 44.6 | 19.4 | 4.6 | 0.8 | 0.1 |
| Dino 75k | 0.5 | 29.1 | 46.1 | 19.4 | 4.2 | 0.6 | 0.1 |

## 4.1 Applications

In this section we present some special applications of the Neural Meshes. The first, of course, is the surface reconstruction from very large point clouds. At the last page we show the reconstruction of the David model from 28 million points in an off-core implementation.

Other possible applications can come by assigning non-uniform probability distributions on a point cloud, obtaining this way an adaptive meshing of it. That means that neural meshes are particularly suitable for reconstructions from point clouds coming from range images, where each point is assigned a confidence value.

If the underlying set $\Omega$ of $\mathscr{P}$ is a triangle mesh instead of a point cloud, we can sample it in two steps by first choosing a triangle and then a point on the chosen triangle. There are many ways to sample the discrete set of triangles, each one giving a different remeshing of the initial mesh. For example, if we sample it uniformly, the probability distribution on the mesh surface is inversely analogous to the triangle's area, and we get a uniform area remeshing. If the probability of each triangle depends on a local estimation of the curvature, then we get a curvature adaptive remeshing. Fig. 8 shows some examples with non-uniform sampling of implicit surfaces.

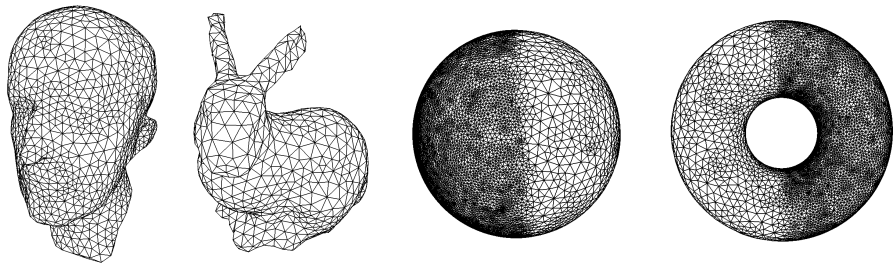


Figure 8: Wireframe views of Neural Meshes.

# 5 Conclusion and Future Work

We presented and analyzed a new method for surface reconstruction based on the principles of statistical learning. The main difference between our method and the majority of the existing methods is that we do not process directly the given data set. Instead we start with a simple mesh which is processed, adapting its geometry and connectivity to the random samples from the data set.

In the future we plan to apply statistical learning methods and develop algorithms for other Computer Graphics related problems. We also plan a more detailed study of the algorithm presented here, because, as we saw above, a deeper theoretical understanding of the algorithm will greatly facilitate its further development.

We think that neural networks and statistical learning methods have a great potential in almost all the Geometric Modeling and Visualization problems and we believe that the present paper offers only a glimpse of that potential.

## Acknowledgments

We thank Professor Marc Levoy for the permission to use the David model.

# References

[1] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi–based surface reconstruction algorithm. In *SIGGRAPH 98, Conference Proceedings*, pages 415–422, 1998.

[2] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH 95, Conference Proceedings*, pages 109–118, 1995.

[3] J. Barhak and A. Fischer. Adaptive reconstruction of freeform objects with 3D SOM neural network grids. In *Pacific Graphics 2001, Conference Proceedings. IEEE Comput. Soc, Los Alamitos, CA, USA*, pages 97–105, 2001.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5] Christian-Arved Bohn. *Radiosity on Evolving Networks*. PhD thesis, Fachbereich Informatik, Universitat Dortmund, Dortmund, Germany, 2000.

[6] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. Richard Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH 01, Conference Proceedings*, pages 67–76, 2001.

[7] B. Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. Technical Report ICSTR-93-026, International Computer Science Institute, Berkeley, 1993.

[8] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[9] B. Fritzke. Growing self-organizing networks – why? In *ESANN'96: European Symposium on Artificial Neural Networks*, pages 61–72, 1996.

[10] J. Giesen and M. John. Surface reconstruction based on a dynamical system. *Computer Graphics Forum (Proceedings of Eurographics '02)*, 21(3):363–371, 2002.

[11] M. Gross and F. Seibert. Visualization of multidimensional data sets using a neural network. *The Visual Computer*, 10(3):145–159, 1993.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning. Data mining, inference, and prediction.* Springer Series in Statistics. New York, NY: Springer., 2001.

[13] M. Hoffmann and L. Várady. Free-form modelling surfaces for scattered data by neural networks. *Journal for Geometry and Graphics*, 1:1–6, 1998.

[14] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH 92, Conference Proceedings*, pages 71–78, 1992.

[15] I. Ivrissimtzis, W-K. Jeong, and H-P. Seidel. Using growing cell structures for surface reconstruction. In *Shape Modeling International 03, Conference Proceedings, (to appear)*, 2003.

[16] L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, 18(3):119–130, 1999.

[17] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[18] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96, Conference Proceedings*, pages 313–324, 1996.

[19] Thomas Martinetz and Klaus Schulten. Topology representing networks. *Neural Networks*, 7(2), 1994.

[20] Alex Pentland and Stan Sclaroff. Closed-form solutions for physically based shape modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):715–729, 1991.

[21] G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH 95, Conference Proceedings*, pages 351–358, 1995.

[22] M. Teichmann and M. Capps. Surface reconstruction with anisotropic density–scaled alpha shapes. In *IEEE Visualization 98, Conference Proceedings*, pages 67–72, 1998.

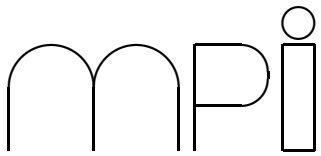
[23] D. Terzopoulos. Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:413–424, 1986.

[24] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH 87, Conference Proceedings*, pages 205–214, 1987.

[25] L. Várady, M. Hoffmann, and E. Kovács. Improved free-form modelling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics*, 3:177–181, 1999.

[26] Yizhou Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization 99, Conference Proceedings*, pages 61–64, 1999.

Figure 9: The Bunny, Eight, Hand, Dino and David reconstructed from 35k, 49k, 136k, 225k, 28m points, at resolution [100,500,1k,5k,20k], [1k,3k,5k,10k,15k], [500,2k,5k,20k,75k], [500,2k,5k,20k,75k], [1k,10k,17k,25k,100k], respectively.




Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

---

| | | |
|---|---|---|
| MPI-I-2003-NWG2-002 | F. Eisenbrand | Fast integer programming in fixed dimension |
| MPI-I-2003-NWG2-001 | L.S. Chandran, C.R. Subramanian | Girth and Treewidth |
| MPI-I-2003-4-008 | C. Rssl, I. Ivrissimtzis, H. Seidel | Tree-based triangle mesh connectivity encoding |
| MPI-I-2003-4-007 | I. Ivrissimtzis, W. Jeong, H. Seidel | Neural Meshes: Statistical Learning Methods in Surface Reconstruction |
| MPI-I-2003-4-006 | C. Rssl, F. Zeilfelder, G. Nrnberger, H. Seidel | Visualization of Volume Data with Quadratic Super Splines |
| MPI-I-2003-4-005 | T. Hangelbroek, G. Nrnberger, C. Rssl, H.S. Seidel, F. Zeilfelder | The Dimension of $C^1$ Splines of Arbitrary Degree on a Tetrahedral Partition |
| MPI-I-2003-4-004 | P. Bekaert, P. Slusallek, R. Cools, V. Havran, H. Seidel | A custom designed density estimation method for light transport |
| MPI-I-2003-4-003 | R. Zayer, C. Rssl, H. Seidel | Convex Boundary Angle Based Flattening |
| MPI-I-2003-4-002 | C. Theobalt, M. Li, M. Magnor, H. Seidel | A Flexible and Versatile Studio for Synchronized Multi-view Video Recording |
| MPI-I-2003-4-001 | M. Tarini, H.P.A. Lensch, M. Goesele, H. Seidel | 3D Acquisition of Mirroring Objects |
| MPI-I-2003-2-003 | Y. Kazakov, H. Nivelle | Subsumption of concepts in $DL\,\mathscr{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete |
| MPI-I-2003-2-002 | M. Jaeger | A Representation Theorem and Applications to Measure Selection and Noninformative Priors |
| MPI-I-2003-2-001 | P. Maier | Compositional Circular Assume-Guarantee Rules Cannot Be Sound And Complete |
| MPI-I-2003-1-011 | P. Krysta, A. Czumaj, B. Voecking | Selfish Traffic Allocation for Server Farms |
| MPI-I-2003-1-010 | H. Tamaki | A linear time heuristic for the branch-decomposition of planar graphs |
| MPI-I-2003-1-009 | B. Csaba | On the Bollobás – Eldridge conjecture for bipartite graphs |
| MPI-I-2003-1-008 | P. Sanders | Soon to be published |
| MPI-I-2003-1-007 | H. Tamaki | Alternating cycles contribution: a strategy of tour-merging for the traveling salesman problem |
| MPI-I-2003-1-006 | M. Dietzfelbinger, H. Tamaki | On the probability of rendezvous in graphs |
| MPI-I-2003-1-005 | M. Dietzfelbinger, P. Woelfel | Almost Random Graphs with Simple Hash Functions |
| MPI-I-2003-1-004 | E. Althaus, T. Polzin, S.V. Daneshmand | Improving Linear Programming Approaches for the Steiner Tree Problem |
| MPI-I-2003-1-003 | R. Beier, B. Vcking | Random Knapsack in Expected Polynomial Time |

| | | |
|---|---|---|
| MPI-I-2003-1-002 | P. Krysta, P. Sanders, B. Vcking | Scheduling and Traffic Allocation for Tasks with Bounded Splittability |
| MPI-I-2003-1-001 | P. Sanders, R. Dementiev | Asynchronous Parallel Disk Sorting |
| MPI-I-2002-4-002 | F. Drago, W. Martens, K. Myszkowski, H. Seidel | Perceptual Evaluation of Tone Mapping Operators with Regard to Similarity and Preference |
| MPI-I-2002-4-001 | M. Goesele, J. Kautz, J. Lang, H.P.A. Lensch, H. Seidel | Tutorial Notes ACM SM 02 A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models |
| MPI-I-2002-2-008 | W. Charatonik, J. Talbot | Atomic Set Constraints with Projection |
| MPI-I-2002-2-007 | W. Charatonik, H. Ganzinger | Symposium on the Effectiveness of Logic in Computer Science in Honour of Moshe Vardi |
| MPI-I-2002-1-008 | P. Sanders, J.L. Trff | The Factor Algorithm for All-to-all Communication on Clusters of SMP Nodes |
| MPI-I-2002-1-005 | M. Hoefer | Performance of heuristic and approximation algorithms for the uncapacitated facility location problem |
| MPI-I-2002-1-004 | S. Hert, T. Polzin, L. Kettner, G. Schfer | Exp Lab A Tool Set for Computational Experiments |
| MPI-I-2002-1-003 | I. Katriel, P. Sanders, J.L. Trff | A Practical Minimum Scanning Tree Algorithm Using the Cycle Property |
| MPI-I-2002-1-002 | F. Grandoni | Incrementally maintaining the number of l-cliques |
| MPI-I-2002-1-001 | T. Polzin, S. Vahdati | Using (sub)graphs of small width for solving the Steiner problem |
| MPI-I-2001-4-005 | H.P.A. Lensch, M. Goesele, H. Seidel | A Framework for the Acquisition, Processing and Interactive Display of High Quality 3D Models |
| MPI-I-2001-4-004 | S.W. Choi, H. Seidel | Linear One-sided Stability of MAT for Weakly Injective Domain |
| MPI-I-2001-4-003 | K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel | Efficient Light Transport Using Precomputed Visibility |
| MPI-I-2001-4-002 | H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel | A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web |
| MPI-I-2001-4-001 | H.P.A. Lensch, J. Kautz, M. Goesele, W. Heidrich, H. Seidel | Image-Based Reconstruction of Spatially Varying Materials |
| MPI-I-2001-2-006 | H. Nivelle, S. Schulz | Proceeding of the Second International Workshop of the Implementation of Logics |
| MPI-I-2001-2-005 | V. Sofronie-Stokkermans | Resolution-based decision procedures for the universal theory of some classes of distributive lattices with operators |
| MPI-I-2001-2-004 | H. de Nivelle | Translation of Resolution Proofs into Higher Order Natural Deduction using Type Theory |
| MPI-I-2001-2-003 | S. Vorobyov | Experiments with Iterative Improvement Algorithms on Completely Unimodel Hypercubes |
| MPI-I-2001-2-002 | P. Maier | A Set-Theoretic Framework for Assume-Guarantee Reasoning |
| MPI-I-2001-2-001 | U. Waldmann | Superposition and Chaining for Totally Ordered Divisible Abelian Groups |
| MPI-I-2001-1-007 | T. Polzin, S. Vahdati | Extending Reduction Techniques for the Steiner Tree Problem: A Combination of Alternative-and Bound-Based Approaches |
| MPI-I-2001-1-006 | T. Polzin, S. Vahdati | Partitioning Techniques for the Steiner Problem |
| MPI-I-2001-1-005 | T. Polzin, S. Vahdati | On Steiner Trees and Minimum Spanning Trees in Hypergraphs |
| MPI-I-2001-1-004 | S. Hert, M. Hoffmann, L. Kettner, S. Pion, M. Seel | An Adaptable and Extensible Geometry Kernel |
| MPI-I-2001-1-003 | M. Seel | Implementation of Planar Nef Polyhedra |
| MPI-I-2001-1-002 | U. Meyer | Directed Single-Source Shortest-Paths in Linear Average-Case Time |
| MPI-I-2001-1-001 | P. Krysta | Approximating Minimum Size 1,2-Connected Networks |
| MPI-I-2000-4-003 | S.W. Choi, H. Seidel | Hyperbolic Hausdorff Distance for Medial Axis Transform |
| MPI-I-2000-4-002 | L.P. Kobbelt, S. Bischoff, K. Khler, R. Schneider, M. Botsch, C. Rssl, J. Vorsatz | Geometric Modeling Based on Polygonal Meshes |
| MPI-I-2000-4-001 | J. Kautz, W. Heidrich, K. Daubert | Bump Map Shadows for OpenGL Rendering |