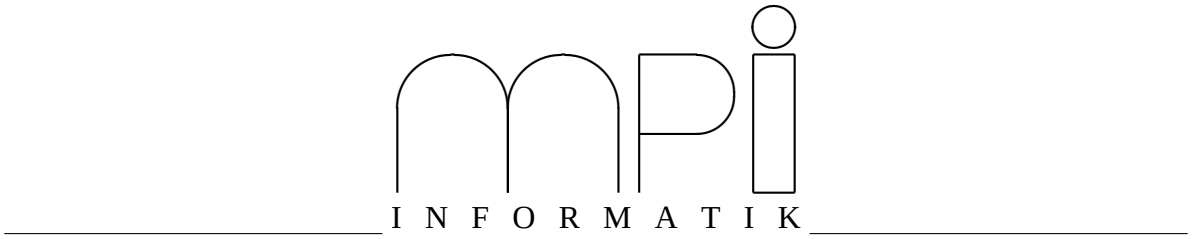

Neural Meshes: Surface Reconstruction with a Learning Algorithm

I.P. Ivrissimtzis W.-K. Jeong S. Lee
Y. Lee H.-P. Seidel

MPI–I–2004–4–005 October 2004

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

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
School of Computing
University of Utah
50 S. Central Campus Dr. Rm. 3190
Salt Lake City, UT 84112, USA
wkjeong@cs.utah.edu

Seungyong Lee
Dept. of Computer Sci. & Eng.
Pohang Univ. of Sci. & Tech. (POSTECH)
Pohang, 790-784, South Korea
leesy@postech.ac.kr

Yunjin Lee
Dept. of Computer Sci. & Eng.
Pohang Univ. of Sci. & Tech. (POSTECH)
Pohang, 790-784, South Korea
jin@postech.ac.kr

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

Abstract

In this paper we propose a Learning algorithm for surface reconstruction. Our algorithm simulates an incrementally expanding neural network which learns a point cloud through a competitive learning process. The topology is learned through statistics based operators which create boundaries and merge them to create handles. We study the algorithm theoretically, analyzing statistically its main components, and experimentally, using an extensive range of input data sets.

Keywords

Neural Networks, Growing Cell Structures, Surface Reconstruction

1 Introduction

Learning algorithms, considered until very recently an experimental technology, have now found commercial application in a wide range of areas, from modeling gene sequences to handwriting recognition. The main reason behind their ever increasing popularity is the need for robust algorithms, able to process very large and/or highly noisy data. In computer graphics, in particular, the problem of processing point clouds obtained from 3D scanners is especially well-suited for Learning methods. Nevertheless, the majority of the existing methods is based more on geometry and less on statistics.

In a typical application of Learning algorithms, we have a data set, which is the result of observation or experiment, and we want to find a mathematical model describing it. In the setting of surface reconstruction, the data set will usually be a point cloud, but can also be an implicit surface or even a triangle mesh, while the generated model will always be a triangle mesh. The main characteristic of the algorithm is that it does not process directly the input point set, but only samples it and uses the sampled points as training data for an incrementally expanding Neural Network with triangle mesh connectivity. We will call the triangle meshes we generate with this method *Neural Meshes*.

1.1 Related Work

The roots of the Learning algorithm we present in this paper go back to the now classic paper [Koh82], where the concept of Self-Organizing Maps (SOM's) was introduced. In [Fri93] the Growing Cell Structures (GSC's) were introduced as a special type of SOM's with the very distinctive feature of growing incrementally, vertex by vertex. In many applications, the feature makes a crucial difference [Fri96].

SOM's and GCS's have already found many applications in geometric modeling and visualization problems. In [GS93], SOM's are used for the visualization of multi-dimensional data. In [HV98], SOM's are used for free-form surface reconstruction and Várady et al. [VHK99] use GCS's for the same purpose. SOM's have also been used for grid fitting [BF01] and surface reconstruction [Yu99]. The main difference between our approach and [Yu99] is that the type of Neural Network we use here grows incrementally. The difference offers the necessary flexibility for learning concavities and other surface features in surface reconstruction.

In [IJS03, JIS03], GCS's are used for surface reconstruction. In [IJS03], a mesh learns a point cloud by adapting the position of its vertices according to signals from the point cloud. The mesh expands by splitting its most active vertices and removing the least active. In [JIS03], curvature adaptive reconstructions are obtained by splitting (removing) the vertices with the most (least) active normals.

Like other Neural Networks applications our method has many parallels with Physics based methods, in particular with the snakes and the active surfaces [Ter86, TPBF87, PS91]. Indeed, a Neural Mesh learns the geometry of the data set by repeated application of a simple step where a vertex moves towards a sample and then its neighborhood is smoothed. The movement towards the sample and the smoothing can be interpreted as an external and an internal force applied on the mesh.

On the other side, there are many well-established techniques, proposing innovative solutions to the surface reconstruction problem from a mainly geometric point of view. Mentioning only the work nearest to Computer Graphics: Hoppe et al. [HDD*92] calculate approximating normals and use volumetric methods to construct a triangle mesh. Bajaj et al. [BBX95] use volumetric techniques on α-shapes to produce a piecewise

polynomial surface. Krishnamurthy and Levoy [KL96] fit a B-Spline surface to the data and then calculate detail vector displacements. Amenta et al. [ABK98, ACK01] use 3D Voronoi diagrams. Teichmann and Capps [TC98] use density scaled α-shapes. Kobbelt et al. [KVLS99] simulate the wrapping of a plastic membrane around the object. Carr et al. [CBC*01] interpolate points with normals using polyharmonic RBF's. Giesen and John [GJ02] solve a dynamical system over a distance function obtained from the sample. Ohtake et al. [OBA*03] use weighted piecewise quadratic functions. We compare these deterministic approaches with the probabilistic algorithm presented here in Section 5.

1.2 Contributions

In this paper, we propose a surface reconstruction technique based on GCS's. The basic framework of the technique is similar to the previous work in [IJS03, JIS03]. However, this paper presents the following major improvements:

- We filter the incoming signals to make the algorithm more robust with respect to outlier noise. No filtering was introduced in [IJS03, JIS03].
- We use a numerically stable counter allowing us to distribute more evenly in time the half-edge collapse operations. Previously the frequency of half edge-collapses was decreasing, which slows down the learning speed of the concavities.
- The topology learning allows the reconstruction of surfaces with handles and boundaries. In [IJS03, JIS03], the mesh always remains a genus 0 surface.
- The mathematical analysis of the algorithm gives an insight into its workings.

1.3 Overview

The Neural Mesh algorithm simulates an incrementally expanding Neural Network learning from a set of training data. We start with a simple mesh, and in response to a signal, we find the nearest vertex of the mesh and move it towards it. Then we smooth the neighborhood of this best matching vertex. Other operations like vertex split, half-edge collapse, triangle removal and boundary merging, based on the combination of an evaluation of the history of each vertex and a statistical analysis of the current status of the mesh, ensure that the mesh grows and adapts to the geometry and topology of the target space.

In Section 2 we present the algorithm in detail. In Section 3 we analyze it, explaining heuristically why it works. In Section 4 we present the results and discuss some special applications. We conclude with a brief comparison between Neural Meshes and other proposed methods.

2 Neural Meshes

The input of the algorithm is an unorganized point cloud Ω which is randomly sampled, and an initial mesh $\mathscr{M}$, usually a tetrahedron, which at each step is processed according to the samples. Because of the sampling process, it is convenient to think of the input data as a probability space $\mathscr{P}$ with underlying set Ω.

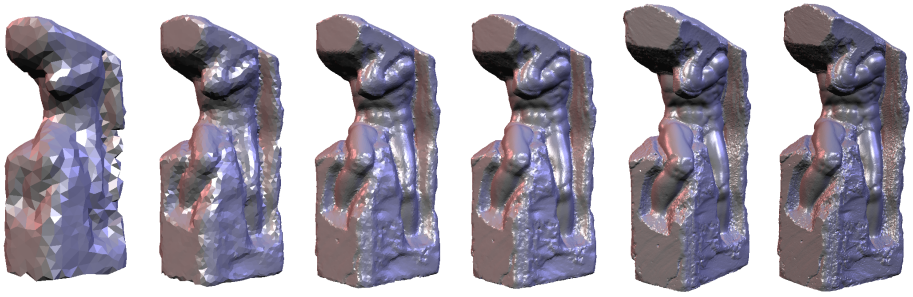

Figure 1: Neural mesh reconstruction of the *Atlas* model with [1K,5K,20K,50K,200K,500K] vertices generated from a data set of four million points.

The algorithm learns the geometry of $\mathscr{P}$ by repeated application of a simple *Basic Step:*
Geometry Learning:

- Sample the target space $\mathscr{P}$ and return one point s.
- Find the vertex v_w of $\mathscr{M}$ which is nearest to s and move it towards s.
- Smooth the 1-ring neighborhood of v_w.

Two less frequent operations expand and optimize the connectivity of $\mathscr{M}$:
Connectivity Change:

- *Vertex Split:* Split the most active vertex.
- *Half-edge Collapse:* Remove the least active vertex.

Finally, two more operations are used for topology learning:
Topology Learning:

- *Triangle Removal:* Remove triangles with extraordinarily large areas.
- *Boundary Merging:* Merge two boundaries that are close to each other.

Next we discuss each step of the algorithm in detail. Notice that the details of the geometry learning and connectivity change steps are changed a lot from [IJS03, JIS03] to handle outliers and to improve the learning speed. The topology learning step is newly introduced in this paper to enable the reconstruction of handles and boundaries of a surface.

2.1 Geometry Learning

2.1.1 Sampling

The first step of the algorithm samples $\mathscr{P}$ returning one point s. In most applications all the points are sampled with equal probability, meaning that $\mathscr{P}$ follows the uniform distribution over Ω. We use a random number generator to obtain independent samples, i.e., with probabilities independent from the previous history of the process. We experimented with more sophisticated sampling methods, trying to avoid as far as possible

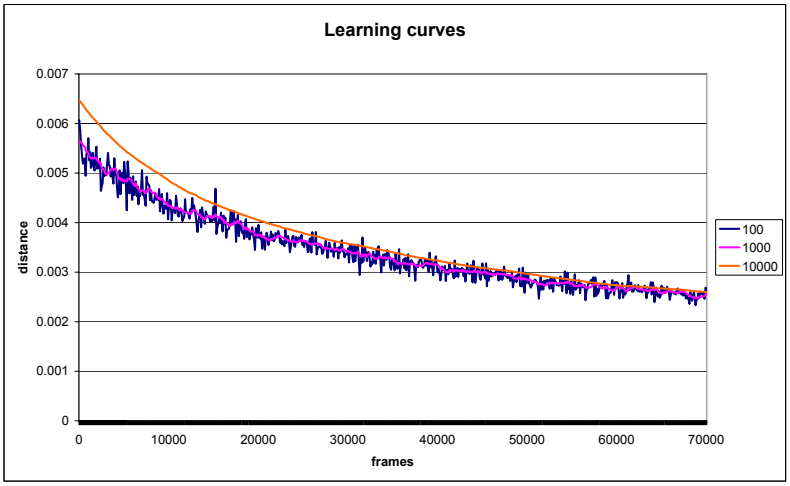

Figure 2: The moving average M_d of the distance between the winner and the sample.

repetitive sampling of the same point, but we saw no improvement, probably because of the large point sets we use.

2.1.2 Update the position of the winner

Next, we find the vertex v_w of $\mathscr{M}$ which is nearest to s. The search uses an octree updated every time the position of a vertex changes. Traditionally, v_w is called the *winner*.

Our main consideration in this step is to avoid the distortion of $\mathscr{M}$ by outlier noise. Assuming that $\mathscr{M}$ is already a good approximation of Ω, we do this by filtering out the outliers in the distribution of distances between samples and winners. Thus, let $\vec{d}$ be the vector $\overrightarrow{v_w s}$ from the winner to the sample. We calculate the moving average M_d and standard deviation σ_d of $|\vec{d}|$. Fig. 2 shows M_d for different windows. Using a window of 10^3 iterations we get a relatively smooth curve, while for a window of 10^4 we get a classic Rate of Learning Curve. In the implementation we use a window of 10^3.

Then, v_w learns from s by moving towards it by

$$\alpha_w \cdot F(|\vec{d}|) \cdot \vec{d} \tag{1}$$

where α_w is a constant controlling the learning pace, and F a function filtering the effects of outliers. Let

$$\varepsilon_t = M_d + \alpha_f \sigma_d \tag{2}$$

be the threshold, controlled with the use of the parameter α_f, above which we want to

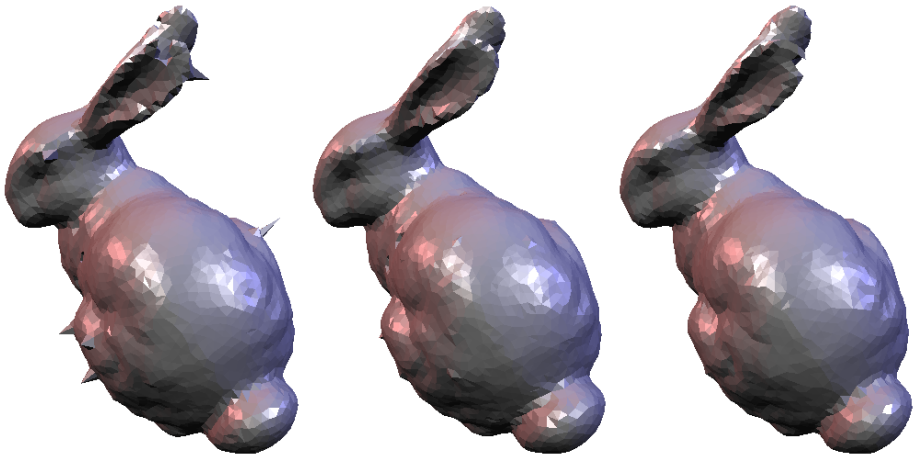

Figure 3: Filtering out a 0.3% of outlier noise. $\alpha_f = 10^6,\ 1,\ 0$.

filter out outliers. As filtering function we used an one-sided variant of Huber's filter

$$F(|\vec{d}|) = \begin{cases} 1 & \text{if} \quad |\vec{d}| \leq \varepsilon_t \\ \varepsilon_t / |\vec{d}| & \text{if} \quad |\vec{d}| > \varepsilon_t \end{cases}$$

which is particularly suitable for filtering out outliers, see [BSMH98]. Fig. 3 shows an example with artificial outlier noise.

Notice that Learning algorithms can naturally cope with a certain amount of noise without any filtering of the incoming information. In our case, if the probability for sampling the outliers is very small, any effect they have on the Neural Mesh will be smoothed out by the sheer amount of information learned from the rest of the data.

2.1.3 Smoothing

We follow a smoothing scheme similar to the one proposed in [IJS03]. It consists of a single iteration of Laplacian smoothing, in the tangential direction, on the 1-ring neighborhood of v_w, with parameter α_L. The normal at v_w is computed before rather than after updating of its position by Eq.(1), and we found that this gives a better estimate of the tangent plane near v_w. The new smoothing scheme makes the algorithm much faster with only marginal loss of quality.

2.2 Connectivity Changes

2.2.1 Vertex split

The connectivity of $\mathscr{M}$ expands by splitting the most active vertices which are assumed to be the vertices with the largest role in the representation of $\mathscr{P}$. To measure the activity of a vertex v, we attach to it a *signal counter* C_v. The counter of the winner is incremented by 1, rewarding v_w for its activity, while the counters of all the other vertices are multiplied by a positive number $\alpha < 1$, with the effect of gradually erasing from them the old activity.

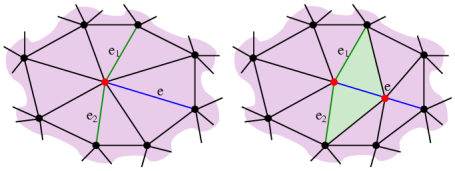

Figure 4: Vertex split.

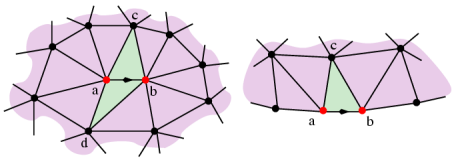

Figure 5: Half edge collapse.

Unlike [Fri93, IJS03], we do not use a constant α but a variable α_V depending on the number V of mesh vertices. To make the parameter α_V more intuitive, we calculate it as a function of the number of iterations $\lambda \cdot V$ required to halve the counter of an inactive vertex, where λ is a constant ($\lambda = 6$ in the implementation). That is, we solve $(\alpha_V)^{\lambda \cdot V} = 1/2$, giving

$$\alpha_V = (1/2)^{1/(\lambda \cdot V)} \tag{3}$$

The α_V controls the pace with which the old information stored in the signal counters is erased. By Eq.(3), the pace at which we erase information slows down when the number of vertices of $\mathscr{M}$ increases and the probability for a vertex to be the winner decreases. As we will see in Section 2.2.2, the way we calculate α_V also bounds the probability of a counter to fall below a threshold, solving some problems of numerical instability arising in [Fri93, IJS03] where $C_v \to 0$ with probability 1 as $V \to \infty$.

To speed up the algorithm we perform a lazy evaluation of the signal counters. Each vertex remembers the number I_v of the last iteration its counter has been updated. If I_c is the number of the current iteration the counter of the winner is updated by

$$C_v := C_v \cdot \alpha_v^{(I_c - I_v)} + 1 \tag{4}$$

When we want to perform a vertex split we update all the signal counters by

$$C_v := C_v \cdot \alpha_v^{(I_c - I_v)} \tag{5}$$

After identifying a vertex v that we want to split, we find the longest edge e with one end at v and traverse both directions from e equally to find two edges e_1, e_2, neighboring v, such that e_1, e_2 split the star of v approximately at half. We split along e_1, e_2 thus distributing the valences as regularly as possible. The new vertex is placed in the middle of e (see Fig. 4).

The counter C_v of the split vertex is divided between the two new vertices in a ratio approximately 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 Section 3. For the implementation details, we follow [Fri93].

The vertex split step is called every C_{vs} iterations of the Basic Step, where C_{vs} is constant, and we split only the vertex with the largest counter. In the implementation we use $C_{vs} = 50$. More iterations per vertex split improve the quality of the reconstruction but increase the computational cost.

2.2.2 Half-Edge Collapse

In a step inverse to the vertex split we remove the underperforming vertices of $\mathscr{M}$ with a half-edge collapse. This step is called after a constant number C_{ec} of vertex splits

and a vertex is removed if it has the lowest counter among the vertices of $\mathscr{M}$, or if its counter is less than $(\alpha_V)^{\mu \cdot V} = (1/2)^{(\mu/\lambda)}$ where μ is a constant. In the implementation $C_{ec} = 5$ and $\mu = 10$.

For a natural interpretation of the constant μ notice that when a vertex is the winner its counter increases above 1. Therefore, it should be inactive for at least $\mu \cdot V$ iterations to fall below the threshold. If all the vertices of $\mathscr{M}$ have equal probability to be winners, then the number x of hits at the next $\mu \cdot V$ iterations follows the Poisson distribution $P(x;\mu)$. In particular, the probability for no hits in the next $\mu \cdot V$ iterations is $P(0;\mu) = e^{-\mu}$. This number is a lower bound for the probability of a counter to be below the threshold when $\mathscr{M}$ is at a stage of equilibrium.

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

$$\frac{1}{3}\sqrt{(a+b-10)^2+(c-7)^2+(d-7)^2} \tag{6}$$

for inner vertices, and

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

for boundary vertices, see Fig. 5.

The half-edge collapse step is instrumental for the quality of the reconstruction and it functions in a many-fold way. First, it removes from $\mathscr{M}$ any misplaced vertices, because such vertices are never the winners and their counter gradually falls below the threshold. In particular, this resolves many situations where $\mathscr{M}$ converges to a local minimum, a common problem with the neural network convergence, and also plays a role in the learning of the concavities of $\mathscr{P}$, (see Fig. 9 Bottom). Another reason for a vertex to be inactive for a long time is that it is located in a part of $\mathscr{M}$ which over-represents $\mathscr{P}$, or in other words, it is located in an area where the competition is very high. By removing such vertices the representation of $\mathscr{P}$ becomes fairer.

The reason for the use of a double criterion for vertex removal is that, on the one hand we want a gradual removal of the underperforming vertices without a large waiting time between two iterations of the step, while on the other hand we want a fast clearing of clusters of underperforming vertices, especially at the early stages of the reconstruction. The former is achieved by removing the vertex with the smallest counter, and the latter by removing the vertices with counters below a threshold.

2.3 Topology Learning

The vertex splits and the half-edge collapses do not change the topology of $\mathscr{M}$. This is done by removing triangles to create boundaries, and by merging these boundaries to create handles. These two steps of the algorithm can be called as often as we wish, but as they are time consuming, it is better to call them rarely and with decreasing frequency. In our implementation we call them every $10 \cdot V$ iterations.

2.3.1 Triangle Removal

As we will see in Section 3, the triangles of $\mathscr{M}$ tend to have area inversely analogous to the probability density of the part of $\mathscr{P}$ they represent. As a result, very large triangles represent parts of $\mathscr{P}$ with very low probability density. When this density falls below a threshold we consider it insignificant and remove the corresponding triangles.

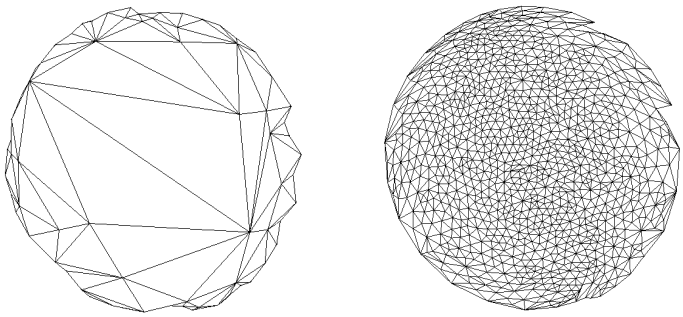

Figure 6: The large triangles on the great circle of a hemisphere are removed, creating a boundary.

A triangle T is considered large and it is removed when

$$\text{Area}(T)/m_A > c_A \tag{8}$$

where m_A is the mean area of the triangles of $\mathscr{M}$ and c_A is a constant. The constant c_A determines the pace with which $\mathscr{M}$ learns the topology. A very large c_A will result in a very slow learning of the topology, while a very small c_A will be prone to inaccurate creation of boundaries. In our implementation we used $c_A = 10$ which gives a good balance between accuracy and learning speed.

A simple calculation shows that the above statistical test is effective. Let B be a planar boundary with perimeter P_B and area A_B. If m_L is the average edge-length of $\mathscr{M}$, then B has an expected number of P_B/m_L vertices. Assuming that all the vertices of $\mathscr{M}$ inside the boundary have been removed with half-edge collapses, the triangles inside B triangulate a polygon, and thus the expected number of triangles inside B is $(P_B/m_L) - 2$. On the other hand, by Heron's formula the expected average area of a triangle of $\mathscr{M}$ is $m_L^2(\sqrt{3}/4)$. Thus, the expected average ratio between a triangle covering B and a triangle of $\mathscr{M}$ is

$$\frac{A_B/(P_B/m_L - 2)}{m_L^2(\sqrt{3}/4)} \sim O(1/m_L) \tag{9}$$

As the Learning progresses and $m_L \to 0$, the ratio in Eq.(9) goes to infinity showing that the boundary is eventually learned. Fig. 6 shows the large triangles on the great disk of the hemisphere that are removed to create a boundary.

2.3.2 Boundary Merge

If two boundaries are relatively near to each other we merge them creating a handle. The criterion for two boundaries to be close enough is that the Hausdorff distance between their vertex sets is below a threshold. In particular, we merge two boundaries b_1, b_2 if

$$H_d(b_1, b_2)/m_L < c_L \tag{10}$$

where $H_d(b_1, b_2)$ is the Hausdorff distance of their vertex sets, m_L is the mean edge-length of $\mathscr{M}$ and c_L is a constant.

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 a triangle is depends on how close it is to being equilateral. Notice that even though the two merging boundaries are close enough, sometimes foldovers may still be created as we connect them. We let these foldovers be resolved later in the process with the Laplacian smoothing of the Basic Step.

For a geometric justification of the criterion in Eq.(10) notice that by definition the Hausdorff distance between the two sets of boundary vertices is a lower bound for the edge-length we will create by merging the two boundaries with a triangle strip. If the Hausdorff distance is too large compared to the average edge of $\mathscr{M}$, the merger of the boundaries would create an outlier edge-length. Therefore it should be rejected.

2.3.3 Connectivity control routines

Further control over the quality of the reconstruction is obtained by imposing several restrictions on the connectivity changing operations. In the implementation we always check the topological legality of the half-edge collapses and triangle removals before executing them, making sure that $\mathscr{M}$ remains manifold throughout the learning process. Also, we do not split a boundary vertex with valence less than or equal to 4 to avoid a proliferation of vertices with negative irregularities on the boundary.

Another possible restriction is to forbid vertex splits and edge collapses that will create vertices of very large valence, or of valence 3. As we will see in Section 3 the algorithm naturally avoids vertices with such valences, but for an absolute guarantee of their absence we have to explicitly forbid any operation creating them.

2.4 Stopping Criteria

The stopping criterion we use here is the number of vertices of $\mathscr{M}$. In certain applications, other stopping criteria controlling the quality rather than the size of the reconstruction may be more suitable. For example we can use the Rate of Learning Curve in Fig. 2 as an approximation of the reconstruction error, and terminate the algorithm when it falls below a threshold. By replacing the distance between the sample and the winner with the distance between the sample and $\mathscr{M}$ we get a more accurate but also more computationally expensive error measure.

3 Analysis

Here we heuristically study the Neural Meshes at the state of equilibrium, that is, when the Learning process has advanced and the mesh is stable. To do this we assume that $\mathscr{M}$ is close enough to $\mathscr{P}$, and thus, $\mathscr{P}$ induces an approximating *probability measure* on $\mathscr{M}$. In the core of the algorithm lies the argument that the density of the vertices of $\mathscr{M}$ is analogous to this probability measure. In other words, we have a larger concentration of vertices near the parts of $\mathscr{P}$ where the distribution of $\mathscr{P}$ is denser.

For a more precise description of the distribution of the vertices of $\mathscr{M}$ we consider their Voronoi diagrams. By definition, a vertex is the winner when the sample is inside

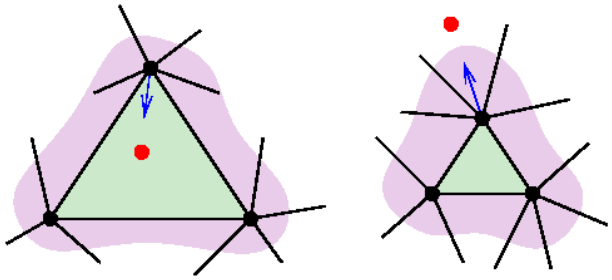

Figure 7: The movement of the winner tends to make triangles with equal probability measure.

its Voronoi cell. That means that the probability of a vertex being the winner is given by the probability measure of its *restricted Voronoi cell*, i.e., the intersection of $\mathscr{P}$ and its Voronoi cell. In equilibrium, the restricted Voronoi cells tend to have equal measure because the vertices with restricted Voronoi cells of large measure are more likely to split, while those with restricted Voronoi cells of small measure are more likely to collapse. That means that the size of the restricted Voronoi cells is inversely proportional to the density of $\mathscr{P}$ and the distribution of the vertices is proportional to that density.

Notice that the above is a global argument. At a local level we may assume that the distribution of $\mathscr{P}$ is uniform, i.e., the probability measure is given by the area. In this case we can argue that the triangles of $\mathscr{M}$ also tend to have equal area. Notice that being at a local level now, the heuristics should also take into account the movement of the vertices at the Basic Step.

Indeed, the movement of the winner towards the sample tends to create triangles of equal area. To see this consider the one ring neighborhood of the winner v_w. If a triangle T in it has an area larger than the average then it is more likely that the sample is in the interior of T which as a result will shrink. On the other hand, if the area of T is small then the sample is more likely to be outside T, which as a result will expand, (see Fig. 7). Finally, the Laplacian smoothing also tends to create triangles of equal area.

The above observation has also implications on the quality of the connectivity of $\mathscr{M}$. To see this notice that the high valence vertices usually have large restricted Voronoi cells and thus, are more likely to split thus improving the connectivity.

The heuristic arguments of this section have been verified experimentally. Fig. 8 shows some typical wireframe views of Neural Meshes. The characteristic semi-regular pattern shown in these two examples is the result of the self-organization of $\mathscr{M}$ during the Learning process. We also notice that the tendency of the vertices to occupy Voronoi cells of equal probability measure leads to meshes with better geometry than that of a random sample from $\mathscr{P}$. Indeed, the expected properties of the Voronoi cells of a uniformly random point set are not so good, see [BEY91].

4 Results

At the end of the paper we show some Neural Mesh reconstructions. The *Youthful* and *Awakening* models in Fig. 10, and the *Atlas* model in Fig. 1, were reconstructed with

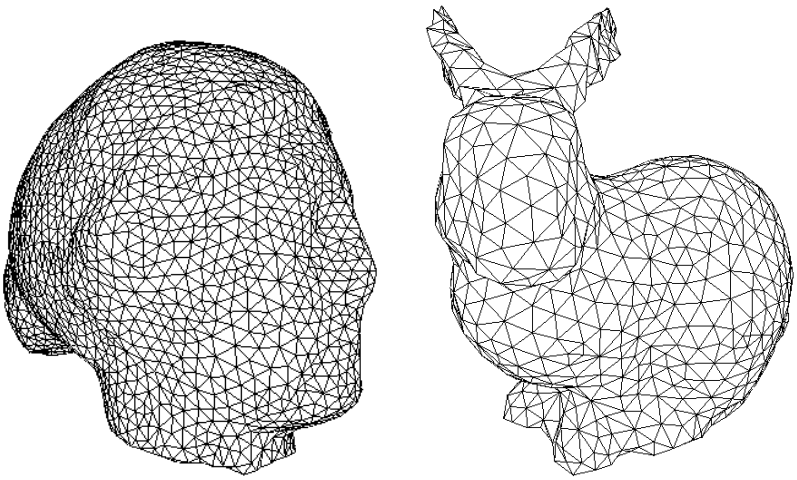

Figure 8: Wireframe views of Neural Meshes.

samples obtained from *VRIP* files in the Digital Michelangelo Archive [LPC*00].

The parameters of the algorithm are very intuitive and can easily be tuned. We run the experiments with the fixed set of parameters shown in Table 1. While tuning them, our main considerations were speed, local smoothness of the Neural Mesh, and the ability to capture the global shape correctly, i.e., avoiding convergence to local minima. The latter is an especially hard problem and different parameters, particular to each model, may work better. Fig. 9 shows a typical behavior that arises when changing one of the parameters.

α_w	α_L	α_f	C_{vs}	C_{ec}	λ	μ	m_A	m_r
0.1	0.05	1	50	10	6	10	10	5

Table 1: Parameters for Neural Meshes

Table 2 shows the timing data for surface reconstruction with Neural Meshes measured on a dual 2.7GHz PC. Notice that the computation time mainly depends on the size of the reconstructed surface. The number of points in the input point cloud has almost no influence on the computation time because random sampling is the only operation to process the point cloud.

Vertices	1K	5K	10K	20K	50K	200K
Time	8s	58s	136s	5m	24m	7h

Table 2: Timing data for Neural Meshes: Since the computation time does not strongly depend on specific models, we only show the timing data with respect to the sizes of the reconstructed surfaces.

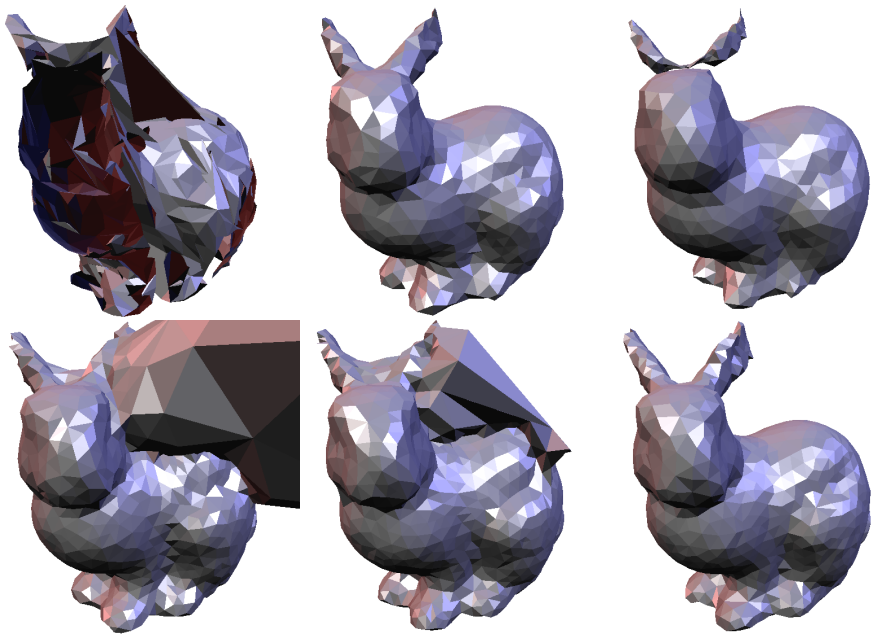

Figure 9: Top: $a_L = 0,\ 0.05,\ 0.5$. Bottom: No half-edge collapses, 1/10 of the default half-edge collapses, default half edge collapses. All the other parameters are default.

5 Discussion

The main characteristics of the Neural Meshes algorithm, compared to the mainstream surface reconstruction algorithms, emanate from its orientation around the concept of Learning. That is, it treats the data as training samples rather than a part of the surface. A first practical consequence is that, as we noticed, the direct processing of the input data is kept to the absolute minimum, and thus, the algorithm copes robustly with very large point clouds. In contrast, geometry oriented algorithms require more extensive processing of the input point cloud, e.g. finding k-neighborhoods [HDD*92], or constructing Voronoi diagrams [ABK98].

The same orientation towards Learning makes it easy to incorporate into the algorithm filters and measurement tools, allowing us to cope with noise and giving us control over the geometry and connectivity of the output. The algorithm copes robustly with topological noise because it uses the whole boundary as the main learning primitive, instead of vertices and edges [MS94]. Also, the reconstruction of the surface is progressive and a first coarse approximation can be obtained immediately.

Another characteristic of the algorithm is that it is probabilistic and the same input can give different outputs. We consider this an advantage, and in the future we plan to exploit the stochastic nature of the algorithm in the form of Neural Mesh *ensémbles*, combining various reconstructions into a single one with better qualities. In contrast, when a deterministic algorithm fails to give a satisfactory reconstruction, we can not improve the result by running the algorithm repeatedly.

One of the limitations of the algorithm is the absence of any guarantee that a sufficiently good sample will result in a correct reconstruction. But notice that the highly noisy output of the 3D scanners makes the existence of such a meaningful guarantee

almost impossible. In practice, we noticed that when many features of the model are close to each other the algorithm can get confused. In particular, when two sheets of the surface are close to each other the algorithm will have difficulties in distinguishing between them.

Another disadvantage of the algorithm is its low speed, compared for example with algorithms for modeling with implicits [OBA*03]. Of course, there is much room for improvement on the timings we report in this paper, especially with regard to the large reconstructions. For example, near the state of convergence, more frequent vertex splits or even global subdivision steps, will accelerate the reconstruction with very little, if any, loss of quality. On the other hand, we believe that Learning algorithms are inherently slower than their non-Learning counterparts, with the extra computations being the price we have to pay for the less detailed instructions we give to the machine.

5.1 Applications

We expect that the Neural Mesh algorithm can simply be extended for several interesting applications:

Out-of-core surface reconstruction: Since the only direct processing of the point cloud is its sampling, we can keep it out of the main memory with only a little timing overhead for sampling. With this approach, we can reconstruct a surface from a very large point cloud, which may not be loaded into the main memory.

Topological simplification: The Neural Mesh algorithm always generates a 2-manifold mesh and in the topology learning step, larger holes are created in the earlier stages of the reconstruction. By using the vertex set of a mesh as the input point cloud, we can simplify the topological complexity of the mesh. This approach can also be used for topological noise removal of a mesh .

Non-uniform meshing: By assigning a non-uniform probability distribution on a point cloud, we can obtain an adaptive meshing of the reconstructed surface. In particular, the probability distribution can reflect the confidence values assigned to the points of a range image [TL94].

5.2 Future Work

In the future we plan a more systematic analysis of the algorithm presented here. A better understanding of the statistics involved will greatly facilitate its further development and we expect that several possible improvements can give Neural Meshes the ability to capture very complicated shapes.

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

References

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi–based surface reconstruction algorithm. In *SIGGRAPH 98, Conference Proceedings* (1998), pp. 415–422.

[ACK01] AMENTA N., CHOI S., KOLLURI R. K.: The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications 19*, 2-3 (2001), 127–153.

[BBX95] BAJAJ C. L., BERNARDINI F., XU G.: Automatic reconstruction of surfaces and scalar fields from 3D scans. In *SIGGRAPH 95, Conference Proceedings* (1995), pp. 109–118.

[BEY91] BERN M. W., EPPSTEIN D., YAO F. F.: The expected extremes in a delaunay triangulation. *Int. J. Computational Geometry and Applications 1*, 1 (1991), 79–92.

[BF01] BARHAK J., FISCHER A.: Adaptive reconstruction of freeform objects with 3D SOM neural network grids. In *Pacific Graphics 01, Conference Proceedings* (2001), pp. 97–105.

[BSMH98] BLACK M. J., SAPIRO G., MARIMONT D., HEEGER D.: Robust anisotropic diffusion. *IEEE Trans. on Image Processing 7*, 3 (1998), 421–432.

[CBC*01] CARR J. C., BEATSON R. K., CHERRIE J. B., MITCHELL T. J., FRIGHT W. R., MCCALLUM B. C., EVANS T. R.: Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH 01, Conference Proceedings* (2001), pp. 67–76.

[Fri93] FRITZKE B.: *Growing Cell Structures - a self-organizing network for unsupervised and supervised learning*. Tech. Rep. ICSTR-93-026, International Computer Science Institute, Berkeley, 1993.

[Fri96] FRITZKE B.: Growing self-organizing networks – why? In *ESANN'96: European Symposium on Artificial Neural Networks, Conference Proceedings* (1996), pp. 61–72.

[GJ02] GIESEN J., JOHN M.: Surface reconstruction based on a dynamical system. *Computer Graphics Forum 21*, 3 (2002), 363–371.

[GS93] GROSS M., SEIBERT F.: Visualization of multidimensional data sets using a neural network. *The Visual Computer 10*, 3 (1993), 145–159.

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *SIGGRAPH 92, Conference Proceedings* (1992), pp. 71–78.

[HV98] HOFFMANN M., VÁRADY L.: Free-form modelling surfaces for scattered data by neural networks. *Journal for Geometry and Graphics 1* (1998), 1–6.

[IJS03] IVRISSIMTZIS I., JEONG W.-K., SEIDEL H.-P.: Using growing cell structures for surface reconstruction. In *Shape Modeling International 03, Conference Proceedings* (2003), pp. 78–86.

[JIS03] JEONG W.-K., IVRISSIMTZIS I., SEIDEL H.-P.: Neural meshes: Statistical learning based on normals. In *Pacific Graphics 03, Conference Proceedings* (2003), pp. 404–408.

[KL96] KRISHNAMURTHY V., LEVOY M.: Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96, Conference Proceedings* (1996), pp. 313–324.

[Koh82] KOHONEN T.: Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43* (1982), 59–69.

[KVLS99] KOBBELT L., VORSATZ J., LABSIK U., SEIDEL H.-P.: A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum 18*, 3 (1999), 119–130.

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3D scanning of large statues. In *SIGGRAPH 00, Conference Proceedings* (2000), pp. 131–144.

[MS94] MARTINETZ T., SCHULTEN K.: Topology representing networks. *Neural Networks 7*, 2 (1994).

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Transactions on Graphics 22*, 3 (2003), 463–470.

[PS91] PENTLAND A., SCLAROFF S.: Closed-form solutions for physically based shape modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 7 (1991), 715–729.

[TC98] TEICHMANN M., CAPPS M.: Surface reconstruction with anisotropic density–scaled alpha shapes. In *IEEE Visualization 98, Conference Proceedings* (1998), pp. 67–72.

[Ter86] TERZOPOULOS D.: Regularization of inverse visual problems involving discontinuities. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8* (1986), 413–424.

[TL94] TURK G., LEVOY M.: Zippered polygon meshes from range images. In *SIGGRAPH 94, Conference Proceedings* (1994), pp. 311–318.

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *SIGGRAPH 87, Conference Proceedings* (1987), pp. 205–214.

[VHK99] VÁRADY L., HOFFMANN M., KOVÁCS E.: Improved free-form modelling of scattered data by dynamic neural networks. *Journal for Geometry and Graphics 3* (1999), 177–181.

[Yu99] YU Y.: Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization 99, Conference Proceedings* (1999), pp. 61–64.

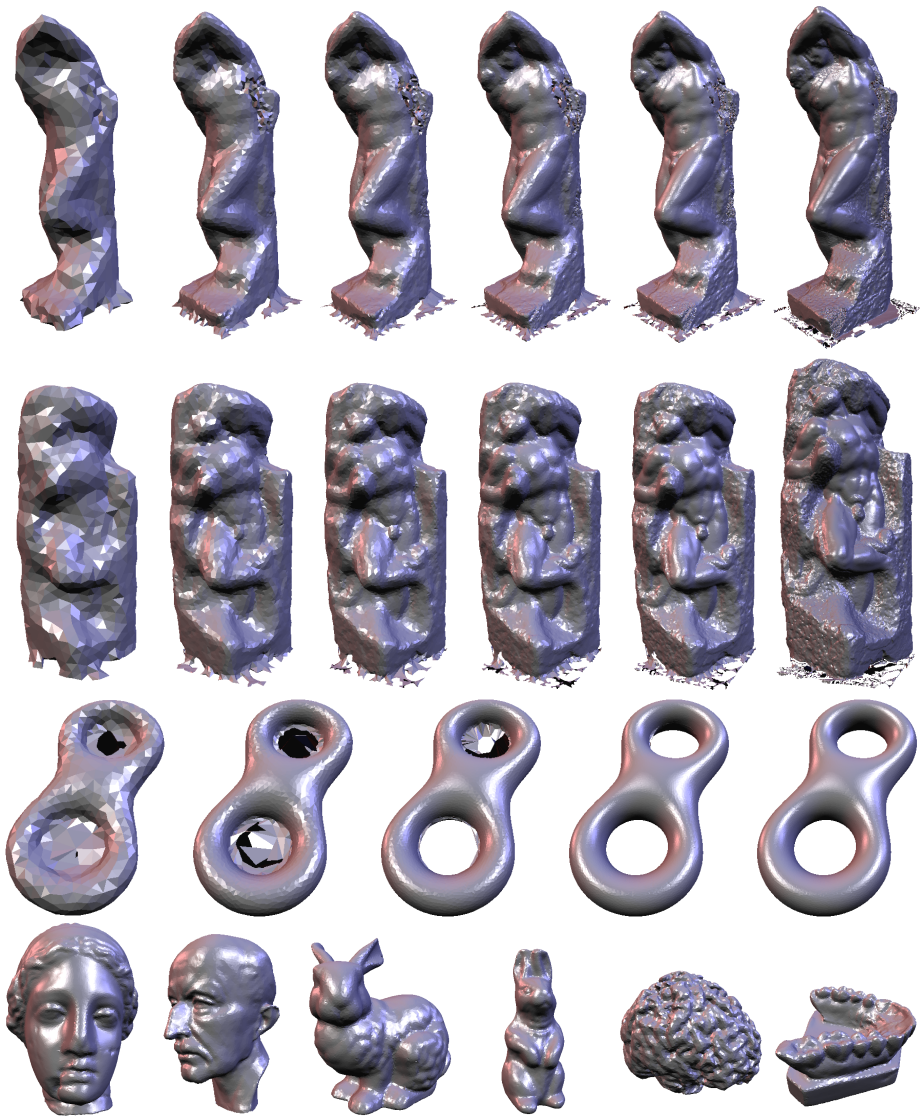

Figure 10: The Youthful, Awakening and the Eight model reconstructed at resolution [1k, 5k, 10k, 20k, 50k, 200k], [1k, 5k, 10k, 20k, 50k, 200k], [1k, 5k, 10k, 20k, 25k] vertices, respectively. The Igea, Max-Planck, Stanford Bunny, Rabbit, Brain, and Teeth model, at resolution [50k, 20k, 20k, 20k, 20k] vertices respectively. The input data sets consisted of 2m, 1.7m, 49k, 134k, 100k, 35k, 67k, 45k, 116k vertices, 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-2004-NWG3-001	M. Magnor	Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae
MPI-I-2004-NWG1-001	B. Blanchet	Automatic Proof of Strong Secrecy for Security Protocols
MPI-I-2004-5-001	S. Siersdorfer, S. Sizov, G. Weikum	Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning
MPI-I-2004-4-004	R. Zayer, C. Rssl, H. Seidel	r-Adaptive Parameterization of Surfaces
MPI-I-2004-4-003	Y. Ohtake, A. Belyaev, H. Seidel	3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs
MPI-I-2004-4-002	Y. Ohtake, A. Belyaev, H. Seidel	Quadric-Based Mesh Reconstruction from Scattered Data
MPI-I-2004-4-001	J. Haber, C. Schmitt, M. Koster, H. Seidel	Modeling Hair using a Wisp Hair Model
MPI-I-2004-2-002	P. Maier	Intuitionistic LTL and a New Characterization of Safety and Liveness
MPI-I-2004-2-001	H.d. Nivelle, Y. Kazakov	Resolution Decision Procedures for the Guarded Fragment with Transitive Guards
MPI-I-2004-1-005	S. Schmitt, L. Fousse	A comparison of polynomial evaluation schemes
MPI-I-2004-1-004	N. Sivadasan, P. Sanders, M. Skutella	Online Scheduling with Bounded Migration
MPI-I-2004-1-003	I. Katriel	On Algorithms for Online Topological Ordering and Sorting
MPI-I-2004-1-002	P. Sanders, S. Pettie	A Simpler Linear Time 2/3 - epsilon Approximation for Maximum Weight Matching
MPI-I-2004-1-001	N. Beldiceanu, I. Katriel, S. Thiel	Filtering algorithms for the Same and UsedBy constraints
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-2003-4-009	N. Zakaria	FaceSketch: An Interface for Sketching and Coloring Cartoon Faces
MPI-I-2003-4-008	C. Roessl, 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. Roessl, F. Zeilfelder, G. Nrnberger, H. Seidel	Visualization of Volume Data with Quadratic Super Splines
MPI-I-2003-4-005	T. Hangelbroek, G. Nrnberger, C. Roessl, 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. Roessl, 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-2003-2-004	A. Podelski, A. Rybalchenko	Software Model Checking of Liveness Properties via Transition Invariants
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-2003-1-018	G. Schaefer	A Note on the Smoothed Complexity of the Single-Source Shortest Path Problem
MPI-I-2003-1-017	G. Schfer, S. Leonardi	Cross-Monotonic Cost Sharing Methods for Connected Facility Location Games
MPI-I-2003-1-016	G. Schfer, N. Sivadasan	Topology Matters: Smoothed Competitive Analysis of Metrical Task Systems
MPI-I-2003-1-015	A. Kovcs	Sum-Multicoloring on Paths
MPI-I-2003-1-014	G. Schfer, L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, T. Vredeveld	Average Case and Smoothed Competitive Analysis of the Multi-Level Feedback Algorithm
MPI-I-2003-1-013	I. Katriel, S. Thiel	Fast Bound Consistency for the Global Cardinality Constraint
MPI-I-2003-1-012		- not published -
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	Polynomial Time Algorithms for Network Information Flow
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