EG UK Theory and Practice of Computer Graphics (2011)
Hamish Carr, Ian Grimstead (Editors)

Memory efficient surface reconstruction based on self organising maps

D. Kaye and I. Ivrissimtzis

Abstract

We propose a memory efficient, scalable surface reconstruction algorithm based on self organising maps (SOMs). Following previous approaches to SOM based implicit surface reconstruction, the proposed SOM has the geometry of a regular grid and is trained with point samples extracted along the normals of the input data. The layer by layer training of the SOM makes the algorithm memory efficient and scalable as at no stage there is need to hold the entire SOM in memory. Experiments show that the proposed algorithm can support the training of the very large SOMs that are needed for richly detailed surface reconstructions.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

The modelling pipeline for creating a 3D model of a real world object starts with the data acquisition. In this stage, a scanning device collects geometric information from the surface of the object. Usually, the obtained data have the form of multiple range scan images, which are then registered and geometric information is extracted. The geometric data takes the form of an unorganised point set, possibly with normals, which can be further processed to remove noise and outliers. Finally, in a process called *surface reconstruction* a surface model of the data is created, usually in the form of a triangular mesh.

One of the most widely used and successful branches of surface reconstruction algorithms is based on implicit surface representations. Implicit methods first compute a scalar field f

$$f : \mathbf{R}^3 \to \mathbf{R} \tag{1}$$

where the value $f(\mathbf{v})$ of a point $\mathbf{v} \in \mathbf{R}^3$ is taken to represent the signed distance between $\mathbf{v}$ and the surface. Under this assumption, the surface is the set of points satisfying $f(\mathbf{v}) = 0$, while f itself is treated as an implicit representation of the surface. Usually, f is computed by solving a global optimisation problem and, its form is typically an aggregate of several implicit functions, each one modelling a small neighborhood of the input point set. Approximate solutions to the global optimisation problem can be efficiently computed, sometimes by just solving a sparse linear system. Finally, a triangular mesh approximation of the surface can be extracted from the zero-level set of f, using an algorithm such as the Marching Cubes [LC87].

The popularity of the implicit methods is mainly due to their robustness. Indeed, implicit methods seem to be particularly well suited to deal with the noisy, unevenly sampled point sets that are the typical outputs of optical scanning devices. Moreover, intensive research activity on implicit methods has yielded some very fast, computationally efficient algorithms.

On the other hand, the extra third dimension of the implicit surface representation may increase the memory requirements of an implicit surface reconstruction algorithm. Memory efficiency problems are dealt with by employing flexible data structures, such as adaptive octrees, however, these complicate the algorithms and increase the implementation overheads. A second drawback of the implicit approach is that the required global optimisation may affect the scalability of the algorithm. Scalability issues are ameliorated by making the locally fitting implicit functions have compact support. However, even though their compact support means that, in principle, the global optimisation problem can be solved locally, in a small neighbourhood of the data, it is nontrivial to implement this in a computationally efficient way.

In this paper we extend the work in [YIL08, KI10], proposing an implicit reconstruction algorithm based on a *self organising map* (SOM). Following [YIL08], the SOM has the connectivity of a 3D regular grid. Its nodes can be

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG11/025-032

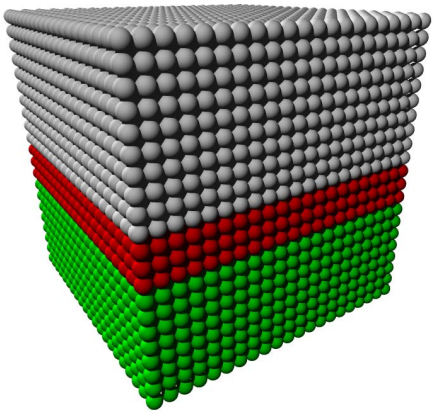

Figure 1: *The SOM is trained layer by layer, starting from the bottom and going up. The already trained nodes are shown in green. The nodes currently being trained are in red. The nodes to be trained are in gray.*

seen as a regular, discrete sample from the inside of a bounding box of the input point set. Each node stores a scalar value representing the signed distance between the node and the surface, and it is trained with data sampled from the normals of the input point set.

The proposed algorithm extends [KI10] by having the dimensions of the SOM adapt themselves to the data provided. Most importantly, the SOM is trained layer by layer, and never stored entirely in memory at any given time, see Fig. 1. Given the ordered, rather than random nature of the training, fewer samples need to be taken from the point cloud to ensure a smooth reconstruction.

Any trained layer can be passed to the Marching Cubes algorithm for polygonisation or saved to disk without needing to wait for the completion of the SOM training. As a result, the algorithm is memory efficient without needing an adaptive data structure, and it is scalable without needing a technically involved localisation of a global optimisation problem. Taking this approach one step further, each layer of the SOM can also be trained at stages, in this case line by line, leading to further memory efficiencies at the expense of higher computational costs.

1.1. Contributions and limitations

The main contributions of the paper are:

- A SOM based, memory efficient, scalable implicit surface reconstruction algorithm.
- A demonstration that the proposed algorithm and a simple modification of the Marching Cubes algorithm can make surface reconstruction memory efficient and scalable, allowing for the reconstruction of very fine triangle mesh representations of the input data.

The main limitations of the paper are:

- a preprocessing of the data, in the form of sorting them according to their z-coordinate is required.
- the difficulty of mathematically modelling the process and thus deriving provable properties of the reconstructed surfaces.

1.2. Overview

The rest of the paper is organized as follows. In Section 2 we briefly review the literature on surface reconstruction. In Section 3 we present the main algorithm of layer by layer implicit SOM training. In Section 4 we present the results of the validation of the algorithm on smooth input data and the testing of the algorithm on raw data. In Section 5 we discuss the implementation of the line by line training of SOM layers and present the results of a proof of concept experiment. We briefly conclude in Section 6.

2. Related work

Over the years, several surface reconstruction algorithms suitable for computer graphics applications have been proposed. [HDD*92, TL94, CL96] are significant examples of early reconstruction algorithms that pioneered the field. In our review of the subsequent research, we first look at two major techniques based on Delaunay tetrahedrisation and moving least squares (MLS) fitting, respectively, then review the implicit and the SOM based methods that are most related to this paper.

Delaunay tetrahedrisation has been repeatedly used for surface reconstruction [ABK98, ACK01, MAVdF05]. Assuming that the faces of the Delaunay tetrahedrisation give a superset of the reconstructed surface, these algorithms use geometrically intuitive criteria to decide which faces are not part of the surface and remove them, leaving a manifold polygonal mesh. Unfortunately, they can be sensitive to poorly sampled data and higher levels of noise.

Algorithms based on the MLS projection form another major branch of surface reconstruction [AK04, RJT*05]. They are the most successful instance of point-set surfaces, where the surface is implicitly defined as the set of the fixed points of a projection, in this case the MLS projection. In a similar approach, [LCOLTE07] use the median projection which does not require local parameterisations of the point set.

Implicit surface reconstruction methods usually require points with normals as inputs [CBC*01, TO02, OBS04]. Using a standard technique, the input data are assigned a scalar value of 0, the ends of the normals pointing to the exterior of the surface are assigned value 1, while the opposite normal ends are assigned value -1. These values are interpolated or approximated by local or global scalar fields, which are then blended to create a single global scalar field approximating

the signed distance to the surface. Finally, the surface is extracted as the zero-level set of the scalar field. Implicit methods perform well when presented with poorly sampled data, but interpolating algorithms are not robust in the presence of noise. Implicit methods also have significant difficulty representing surfaces with boundaries.

Several surface reconstruction algorithms are based on SOMs and their variants. SOMs have been used for grid fitting in [BF01] and for surface reconstruction in [Yu99]. In [Fri93,IJS03], special types of SOMs called *Growing Cell Structures*, that dynamically create edges between nodes, are used for the same problem. Unfortunately, the growing cell structures required the entire point cloud to be sampled several times in order to achieve a stable result.

Surface reconstruction is still a very active research area with many recently proposed algorithms employing a variety of geometric, statistical and signal theoretic techniques. [NRDR05] combines separately acquired positional and normal information. [KMA06] processes dense point sets obtained from multiple scans with a variant of the image processing technique of super-resolution. [KBH06, BKBH09] solve a Poisson equation, which is used for comparison later in the paper.

3. Main Algorithm

In this section we describe the main algorithm and discuss some implementation details. The input of the algorithm is a point set with normals. If the point set has no normals then we can estimate them using some of the standard methods in the literature.

The SOM is arranged in the form of a regular 3D grid with the nodes on the lattice $\mathbf{Z}^3$. That is, each node has integer coordinates and the length of each edge is 1. Each node stores an estimation of its signed distance from the surface, $\bar{d}$. The edges provide no information and can be completely ignored since neighborhood relations for the nodes of the grid are obtained by direct means such as distances between nodes.

The SOM has a band of active layers, in which each node stores a list L of weighted distances from the surface, which is obtained through training, $\bar{d}$ computed as a weighted average of the elements of L and represents the current estimate for the value of the implicit function at that node. Only nodes in this active band are trained. The active band moves from the base to the top of the SOM, training it.

When fully trained, the SOM represents a discrete implicit description of the surface that can be triangulated using the marching cubes algorithm [LC87]. Even though each node only passes its value $\bar{d}$ to the marching cubes algorithm, the list of weighted distances L provides information about earlier states of the SOM, which can be used for fine-tuning or analysis of the results. It also provides some robustness to noisy input data or misaligned normals of the type that are common when processing data from optical scanners.

3.1. Data alignment and sorting

In the first step of the algorithm, we find a tight rectangular bounding box for the input point set and align it to the SOM. Using a standard technique for the efficient computation of a bounding box, we apply Principal Component Analysis on the input point set and use the three principal components as the axes of the box. Next, by an affine transformation followed by scaling we map the bounding box and the data into the convex hull of the SOM grid. In the labelling of the axes we choose the z-axis to be the the largest principal component ensuring that the base of the SOM is as small as possible, affording us the smallest memory requirement. The point cloud is analysed and its maximal and minimal x and y values found. The SOM then configures its dimensions accordingly. Finally, the point set is sorted by z-coordinate, low to high.

3.2. Training Step

The basic training step of the SOM runs as follows:

1. A sample s is extracted from the point set.
2. Training data are created from the sample as shown in Fig. 3.2. The training data extend a distance of ± 2 units from the sample and carry their distance from the sample, and a weight (computed as $1/(1+d_s^2)$) where d_s is the distance from the original sample, s.
   The weight of a training point represents our confidence that its distance from the surface is accurate, this might not be the case if two areas of the surface are close to each other, or if there is another sample point closer to the training point than s.
3. For each training point, the nearest SOM node is found, this node has the weight and distance of the training point added to its list of distances, L.

Samples are extracted sequentially and are assumed to lie on the surface, i.e. we assume that their distance from the surface is 0.

3.3. Smoothing

The estimated signed distance $\bar{d}$ from the surface of every active node is found by computing the weighted average of the elements in its distance list. The nodes to be smoothed are then subjected to the following procedure.

1. Using the natural L_0 metric of the SOM grid, the $L_0(1)$ neighbours of node n are found. If n has no trained neighbours, it is not smoothed. Otherwise, the mean of the $\bar{d}$'s of the trained neighbours, m_1, is computed.
2. Similarly to above, the $L_0(2)$ neighbours are found. If there are less than two trained $L_0(2)$ neighbours then n is

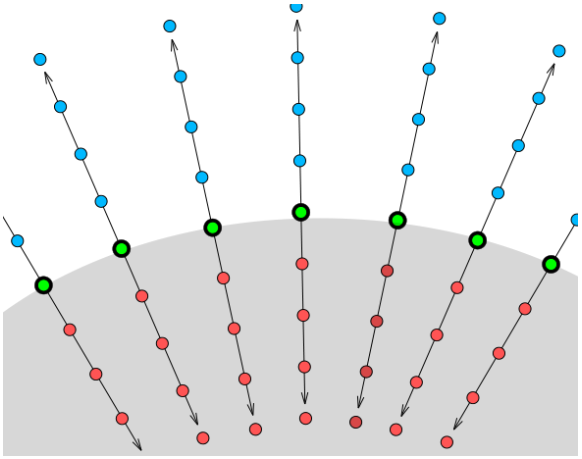

Figure 2: *The grey area represents the interior of the surface. Red points are training data with negative distances from the surface, blue points are those with positive distances, and the green points have distance zero.*

not smoothed. Otherwise, the mean of the trained neighbours, m_2 is computed.

3. Finally, n has the distance $0.65m_1 + 0.35m_2$ added to its distance list with a weight of 1.0. Weights of 0.65 and 0.35 were experimentally determined to work well for a variety of data.

The z-coordinate z_s of the latest sample is stored when the smoothing phase is entered. Upon subsequent sample extractions, the new sample's z-coordinate is compared to this z_s and if the difference is more than a fixed amount, the SOM is smoothed again, and z_s is updated. Initially, z_s holds the z-coordinate of the first sample taken.

The bottom two layers of the active band are not smoothed because the computed distance would not include contributions from the $L_0(1)$ and $L_0(2)$ neighbours with the lowest z-coordinates and so would lead to biased smoothing.

The smoothing does not go all the way to the top of the active band, the z-coordinate of the latest sample point to be extracted, z_s, is used to calculate an upper limit: no node is smoothed if the distance between its layer and z_s is less than 2. Similarly to the restriction at the bottom of the active band: layers very close to the top would be unlikely to have trained upper neighbours, which could lead to biased smoothing.

3.4. Storing

If the z-coordinate of any sample point is within 2 units of the top of the active band then it triggers a dumping of the bottom 2 layers. The data for these layers is saved to a file (or it could be directly passed to the marching cubes algorithm), and the active band moves 2 layers in the positive z direction. The memory for the two formerly active layers is then freed, helping to keep the memory consumption within reasonable bounds.

3.5. Parameter choice

Setting the height of the active band to 20 nodes resulted to good quality surfaces without using large amounts of memory.

The training data extend 2 units from their sample point in the direction of the normal, and in the same distance the opposite direction. This distance was chosen because only nodes close to the surface will have any effect on its geometry when the marching cubes is run. Consequently, training nodes further away would increase the memory requirements for no benefit. This is also the distance to the top of the active band that triggers storing since to have a sample closer than 2 units to the top of the active band would mean its training data extending beyond the top.

Given that training data extend ± 2 units from the sample points, nine training data were created per sample. This was to ensure that nodes were consistently trained but not over-trained. Consider an axis-aligned sample point: creating 9 training data ensures that the inter-point distance is 0.5 units. Training data are applied to the nearest node, and this would make sure that each node is trained exactly once by the sample. A more sparse set of training data would lead to gaps (and thus inconsistent distances) and a denser set would result in each node being trained with multiple inconsistent distances.

The length of the weighted distance list is constrained to provide a bound on the memory that can be used. Each node can store a maximum of 100 weighted distances. This was chosen to be long enough to tolerate noise (because the effect of the other distances dwarfs that of the noise) but short enough to keep memory within sensible limits.

4. Results

At each stage of testing, the proposed algorithm was compared to [KI10] and [KBH06]. It should be noted that [KBH06] produced smoother meshes than the other two algorithms.

We first validate the proposed algorithm by testing it on point sets obtained by stripping the connectivity from smooth meshes. We used the neptune, turbine blade, happy buddha and dragon meshes. By comparing the re-reconstructed meshes with the original meshes, which serve as the ground truth for the underlying surface of the point data, we are able to gauge the accuracy of the method. Figure 3 shows the obtained reconstructions and Figure 4 shows close-ups of the reconstructions.

Next, we tested the surface reconstruction algorithm on

Figure 3: *Re-reconstructions from smooth meshes. The original meshes are on the left, reconstructions on the right.*

unprocessed point sets from raw range scan data, in particular the Bunny data from Stanford repository and the Ramesses data from AIM@SHAPE. The normals for the Ramesses model were computer from the raw mesh provided by AIM@SHAPE (using MeshLab) as the weighted average of incident face normals. Figure. 5 shows the obtained reconstructions. Figure. 6 shows a close-up of the Ramesses reconstruction.

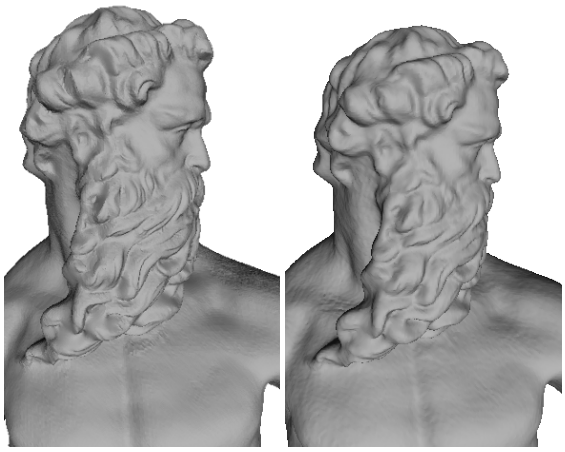

Figure 4: *Close-up of Neptune's face, reconstructed from a mesh.*

To validate the memory efficiency claim, memory consumption was monitored during the reconstruction of the models (whose sizes are shown in Table 1). The average and peak memory use for each model are displayed in Tables 2 and 3 respectively. An "X" in any table indicates that the algorithm was not able to run to completion on a PC with 4GB of RAM. A detailed breakdown of timings for the proposed method is shown in Table 4 and the total run-time for each model and method are shown in Table 5. The marching cubes implementation used was not able to extract the isosurface of the huge neptune model due to insufficient memory. The number of triangles in each model after reconstruction by each method is shown in Table 6.

The neptune model was reconstructed at a variety of scales, with the timing recorded. The results are shown in Figure 7 as a function of the volume of the point cloud's bounding box, or equivalently the number of SOM nodes. All other parameters were kept constant for these reconstructions to ensure that only the scale affected the results. As can be seen, the timing scales almost linearly with the volume of the bounding box.

model	bounding box	points
buddha	$140 \times 122 \times 300$	543'652
dragon	$185 \times 235 \times 299$	437'645
turbine	$495 \times 463 \times 598$	882'954
neptune	$302 \times 694 \times 1001$	2'003'931
huge neptune	$2112 \times 4858 \times 7004$	2'003'931
bunny scans	$130 \times 209 \times 210$	362'272
ramesses	$224 \times 318 \times 645$	826'266

Table 1: *The dimensions of each point cloud's bounding box in the x, y and z directions, along with the number of points in each cloud.*

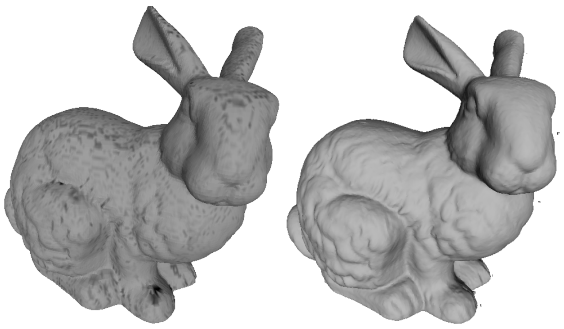

Figure 5: *Reconstruction from scan data. The mesh supplied from the Stanford 3D Scanning Repository is on the left, the reconstruction is on the right.*

Figure 6: *Reconstruction from scan data. The mesh supplied from AIM@SHAPE is on the left, the reconstruction on the right.*

The peak RAM use reported in [OBA*03] when reconstructing the buddha, dragon and bunny scans was 442MB, 210MB and 110MB respectively. Carr *et. al.* reported a peak RAM use of 306MB to reconstruct the dragon and 291MB to reconstruct the buddha from the same point clouds [CBC*01].

5. Line by line SOM training

If further memory efficiency is required then the SOM can be modified to be trained line by line, as shown in Figure 8. After the initial preprocessing and sorting, the points within the range $(z, z+1)$, for integral z, are sorted low to high (left to right) by their y-coordinate. In this case the active band becomes an active line, which has fixed size in both the y and z directions.

model	average (proposed)	average [KI10]	average [KBH06]
buddha	16	933	173
dragon	20	1260	81
turbine	48	X	144
neptune	54	X	205
huge neptune	1633	X	X
bunny scans	18	623	73
ramesses	29	X	19

Table 2: *The average memory use of the reconstructions in Megabytes.*

model	peak (proposed)	peak [KI10]	peak [KBH06]
buddha	23	1209	173
dragon	24	1721	253
turbine	71	X	384
neptune	112	X	312
huge neptune	1772	X	X
bunny scans	20	885	178
ramesses	38	X	90

Table 3: *The peak memory use of the reconstructions in Megabytes.*

5.1. Implementation

Inactive nodes are stored in temporary files, and like the layer by layer reconstruction, cannot be trained. If the z-coordinate of any sample point is too close to the top of the SOM, the SOM dumps these layers and moves in the positive z direction.

Similarly, if the y-coordinate of any sample point is too close to the rightmost edge of the active line, the leftmost 2 rows of nodes are stored in temporary files (including their distance list) and the active linemoves to the right. When the SOM has been trained with all the sample points in a layer (indicated by $y_{n+1} < y_n$) the SOM stores the current state of all active nodes in the temporary files and the active line jumps back to the left.

The temporary files are read to determine the state of the nodes when they were last active. The training then continues as before, but whenever the SOM moves right, it reads the states of the now-active nodes from the temporary files.

Using an active line with a height and width of 30 nodes resulted to good quality surfaces and low memory use. A larger value was used compared to the height of the active band to take into account that smoothing the active line would propagate the training information less than in the active band. Smoothing can be triggered by z-coordinate changes, as in the layer by layer reconstruction, or by y-coordinate changes.

model	preprocessing time (s)	training time (s)	polygonisation time (s)
buddha	12	45	5
dragon	10	38	6
turbine	19	227	40
neptune	50	302	57
huge neptune	51	∼8 hrs	X
bunny scans	9	21	3
ramesses	20	120	19

Table 4: *Timings for the different stages of the proposed algorithm.*

model	recon. (s) proposed	recon. (s) [KI10]	recon. (s) [KBH06]
buddha	62	142	163
dragon	54	159	220
turbine	286	X	366
neptune	409	X	400
bunny scans	33	98	50
ramesses	159	X	39

Table 5: *The total run-time for each algorithm, including any preprocessing and isosurface extraction.*

model	triangles (proposed)	triangles [KI10]	triangles [KBH06]
buddha	182'421	343'501	629'208
dragon	247'751	436'873	856'976
turbine	1'600'242	X	1'359'064
neptune	1'070'264	X	1'403'528
bunny scans	128'884	221'166	211'930
ramesses	577'923	X	111'980

Table 6: *The number of triangles for each method.*

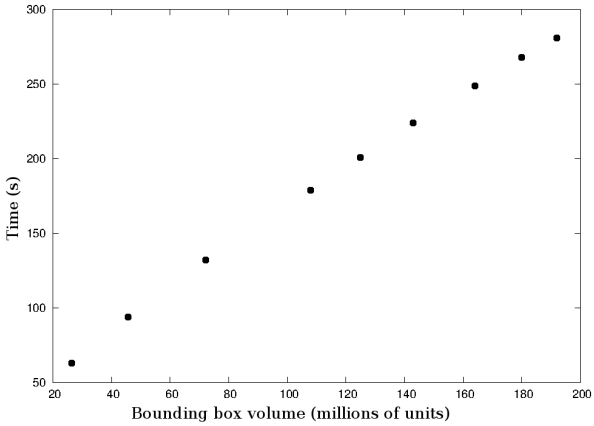

Figure 7: *Time taken to reconstruct the neptune model vs. the volume of its bounding box.*

5.2. Results

The current implementation of the line by line SOM training is basic and cannot handle large input data sets. However, proof of concept results on small data sets show significant memory savings. For example; when the Ramesses model (scaled to $112 \times 159 \times 322$) was reconstructed, the mean and peak memory consumption were only 2.4MB and 2.5MB respectively. On the other hand, the time taken to complete the reconstruction was 44 minutes (148'495 triangles). The reconstructed model is shown in Figure 9.

6. Conclusions

We presented an implicit surface reconstruction algorithm based on self organising maps. The main novelty of our approach is the layer by layer training of the SOM's nodes, which means there is no need to store the entire SOM in memory at any stage. The memory efficiency of the algorithm compared to [CBC*01, OBS04, KBH06] and [KI10] was demonstrated. Good sized SOMs, such as those used for the reconstructions of Neptune and the turbine, require about 100MB peak memory, while even the massive SOM used for reconstructing the huge Neptune model can be accommodated in the memory of a commodity PC.

The second major advantage of our approach is its scalability. Not only can the training of the SOM be done layer by layer, but, in a recursive application of this principle, a layer can be trained line by line. The preliminary results of this line by line training are promising. If further memory efficiency is needed, a line could be trained thick point by thick point, see Fig. 10.

We note that memory efficiency and scalability are natural features of our approach and can be achieved with minimal implementation overheads. In contrast, memory efficiency in other implicit reconstruction methods requires the implementation of complex data structures, such as adaptive octrees, or the use of special scalable algorithms for solving global optimisation problems.

The regular structure of the SOM employed by the proposed algorithm, and the very simple processing operations performed at each node, make the method particularly suitable for GPU implementation. In the future, we plan to work

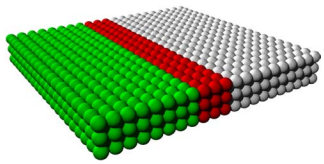

Figure 8: *In a recursive application of the layer by layer training principle, a layer can be trained line by line.*

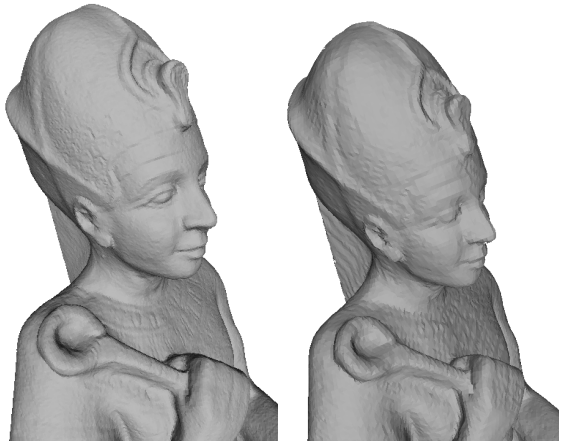

Figure 9: *Line by line reconstruction from scan data. The layer by layer reconstruction is on the left, and the lower-resolution line by line reconstruction on the right.*

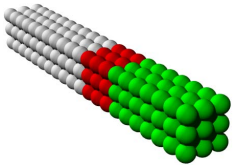

Figure 10: *A line could be trained thick point by thick point.*

on a GPU implementation of the algorithm which, together with existing GPU implementations of the marching cubes algorithm [JC06], could be a step towards the goal of real time surface reconstruction.

Source code is available for [KI10], [KBH06] and the proposed algorithm under a BSD licence.

References

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi–based surface reconstruction algorithm. In *SIGGRAPH* (1998), pp. 415–422. 2

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

[AK04] AMENTA N., KIL Y. J.: Defining point-set surfaces. *ACM Trans. Graph. 23* (2004), 264–270. 2

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

[BKBH09] BOLITHO M., KAZHDAN M., BURNS R., HOPPE H.: Parallel poisson surface reconstruction. In *Advances in Visual Computing*, vol. 5875 of *Lecture Notes in Computer Science*. 2009, pp. 678–689. 3

[CBC*01] CARR J., BEATSON R., CHERRIE J., MITCHELL T., FRIGHT W., MCCALLUM B., EVANS T.: Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH* (2001), pp. 67–76. 2, 6, 7

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *SIGGRAPH* (1996), pp. 303 – 312. 2

[Fri93] FRITZKE B.: *Growing Cell Structures - a self organizing network for unsupervised and supervised learning*. Tech. Rep. ICSTR-93-026, ICSI, Berkeley, 1993. 3

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. In *SIGGRAPH* (1992), pp. 71–78. 2

[IJS03] IVRISSIMTZIS I., JEONG W.-K., SEIDEL H.-P.: Using growing cell structures for surface reconstruction. In *SMI* (2003), pp. 78–86. 3

[JC06] JOHANSSON G., CARR H.: Accelerating marching cubes with graphics hardware. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research* (2006), ACM. 8

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *SGP* (2006), pp. 61–70. 3, 4, 6, 7, 8

[KI10] KAYE D., IVRISSIMTZIS I.: Implicit Surface Reconstruction and Feature Detection with a Learning Algorithm. Collomosse J., Grimstead I., (Eds.), Eurographics Association, pp. 127–130. 1, 2, 4, 6, 7, 8

[KMA06] KIL Y., MEDEROS B., AMENTA N.: Laser scanner super-resolution. In *SoPBG* (2006), pp. 9–16. 3

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3D surface construction algoritm. In *SIGGRAPH* (1987), pp. 163–168. 1, 3

[LCOLTE07] LIPMAN Y., COHEN-OR D., LEVIN D., TAL-EZER H.: Parameterization-free projection for geometry reconstruction. *ACM ToG 26*, 3 (2007), 22. 2

[MAVdF05] MEDEROS B., AMENTA N., VELHO L., DE FIGUEIREDO L. H.: Surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing* (2005), pp. 53–62. 2

[NRDR05] NEHAB D., RUSINKIEWICZ S., DAVIS J., RAMAMOORTHI R.: Efficiently combining positions and normals for precise 3D geometry. In *SIGGRAPH* (2005), pp. 536–543. 3

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. In *SIGGRAPH* (2003), pp. 463–470. 6

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: 3D scattered data approximation with adaptive compactly supported radial basis functions. In *SMI* (2004), pp. 31–39. 2, 7

[RJT*05] REUTER P., JOYOT P., TRUNZLER J., BOUBEKEUR T., SCHLICK C.: Surface reconstruction with enriched reproducing kernel particle approximation. In *SoPBG* (2005), pp. 79–87. 2

[TL94] TURK G., LEVOY M.: Zippered polygon meshes from range images. *SIGGRAPH* (1994), 311–318. 2

[TO02] TURK G., O'BRIEN J.: Modelling with implicit surfaces that interpolate. *ACM ToG 21*, 4 (2002), 885–873. 2

[YIL08] YOON M., IVRISSIMTZIS I., LEE S.: Self-organising maps for implicit surface reconstruction. In *Theory and Practice of Computer Graphics* (2008), EG Press, pp. 83–90. 1

[Yu99] YU Y.: Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization* (1999), pp. 61–64. 3